|  |
| --- |
| Kingswood School |
| Survival of the fittest teaching aid |
| Comp4 project |

|  |
| --- |
| Alex Robinson |

[Section 1: Analysis 6](#_Toc383631628)

[The problem: identification and background 6](#_Toc383631629)

[Description of available tools 7](#_Toc383631630)

[Identification of prospective users 9](#_Toc383631631)

[Identification of user needs and acceptable limitations 10](#_Toc383631632)

[Data sources and destinations 11](#_Toc383631633)

[Data volumes 11](#_Toc383631634)

[Analysis data dictionary 12](#_Toc383631635)

[Structure and data flow diagrams 12](#_Toc383631636)

[Entity-relationship (E-R) model 12](#_Toc383631637)

[Object analysis diagram 13](#_Toc383631638)

[Objectives for the proposed system 13](#_Toc383631639)

[Appraisal of potential solutions 15](#_Toc383631640)

[Proposed solution 16](#_Toc383631641)

[Justification of chosen solution 17](#_Toc383631642)

[Section 2: Design 19](#_Toc383631643)

[Overall system design 19](#_Toc383631644)

[Description of modular structure of system 21](#_Toc383631645)

[Definition of data requirements (Design Data Dictionary) 31](#_Toc383631646)

[Description of record structure 33](#_Toc383631647)

[File organisation and processing 34](#_Toc383631648)

[Validation required 34](#_Toc383631649)

[Identification of storage media 34](#_Toc383631650)

[Algorithms 35](#_Toc383631651)

[Class definitions (diagrams) and details of object behaviours and methods 45](#_Toc383631652)

[User interface design (HCI) rational 47](#_Toc383631653)

[UI sample of planned data capture and entry designs 53](#_Toc383631654)

[UI sample of planned valid output designs 53](#_Toc383631655)

[Description of measures planned for security and integrity of data 56](#_Toc383631656)

[Overall test strategy 56](#_Toc383631657)

[Section 3: Technical Solution 57](#_Toc383631658)

[Section 4: System Testing 58](#_Toc383631659)

[Test Plan 58](#_Toc383631660)

[Test data 59](#_Toc383631661)

[Section 5: System Maintenance 146](#_Toc383631662)

[System overview 146](#_Toc383631663)

[Modular structure overview 146](#_Toc383631664)

[Classes 154](#_Toc383631665)

[Modules 167](#_Toc383631666)

[Configuration 190](#_Toc383631667)

[Test driven development 192](#_Toc383631668)

[Section 6: User Manual 195](#_Toc383631669)

[Introduction 196](#_Toc383631670)

[Simulation overview 196](#_Toc383631671)

[Installation 199](#_Toc383631672)

[Layout 200](#_Toc383631673)

[Use 202](#_Toc383631674)

[Errors 208](#_Toc383631675)

[Section 7: Evaluation 214](#_Toc383631676)

[Evaluation 214](#_Toc383631677)

[User feedback 217](#_Toc383631678)

[Analysis of feedback 217](#_Toc383631679)

[Suggestions and improvements 219](#_Toc383631680)

[Appendix A: Interviews 221](#_Toc383631681)

[Appendix B: Evolution – The main concepts 226](#_Toc383631682)

[Appendix C: Source code 227](#_Toc383631683)

[index.html 227](#_Toc383631684)

[style.css 229](#_Toc383631685)

[core/Ant.js 231](#_Toc383631686)

[core/Worker.js 237](#_Toc383631687)

[core/Soldier.js 241](#_Toc383631688)

[core/Queen.js 247](#_Toc383631689)

[core/Species.js 250](#_Toc383631690)

[core/Nest.js 253](#_Toc383631691)

[core/NestPiece.js 257](#_Toc383631692)

[core/Pheromone.js 259](#_Toc383631693)

[core/FoodSystem.js 260](#_Toc383631694)

[core/Food.js 262](#_Toc383631695)

[core/map.js 264](#_Toc383631696)

[core/controls.js 265](#_Toc383631697)

[core/canvas.js 274](#_Toc383631698)

[core/utilities.js 276](#_Toc383631699)

[core/config.js 282](#_Toc383631700)

[core/main.js 290](#_Toc383631701)

[Appendix D: Test source code 295](#_Toc383631702)

[Ant.scan.test 296](#_Toc383631703)

[Ant.secrete.test 299](#_Toc383631704)

[Ant.seeNest.test 301](#_Toc383631705)

[Ant.smell.test 303](#_Toc383631706)

[Ant.takeFood.test 304](#_Toc383631707)

[Ant.wonder.test 307](#_Toc383631708)

[Food.draw.test 310](#_Toc383631709)

[FoodSystem.addFood.test 311](#_Toc383631710)

[FoodSystem.addFoodBlob.test 313](#_Toc383631711)

[Nest.calcSpeciesCost.test 316](#_Toc383631712)

[Nest.reproduce.test 317](#_Toc383631713)

[NestPiece.draw.test 320](#_Toc383631714)

[Pheromone.draw.test 321](#_Toc383631715)

[Queen.draw.test 323](#_Toc383631716)

[Soldier.attack.test 324](#_Toc383631717)

[Soldier.draw.test 326](#_Toc383631718)

[Soldier.guardFood.test 327](#_Toc383631719)

[Soldier.guardNest.test 329](#_Toc383631720)

[Soldier.soldiersInView.test 331](#_Toc383631721)

[angleTo.test 335](#_Toc383631722)

[boundary.test 339](#_Toc383631723)

[clone.test 340](#_Toc383631724)

[coordToIndex.test 342](#_Toc383631725)

[distance.test 342](#_Toc383631726)

[getBlock.test 343](#_Toc383631727)

[getCellCoord.test 345](#_Toc383631728)

[getSector.test 347](#_Toc383631729)

[indexToCoord.test 350](#_Toc383631730)

[randColour.test 351](#_Toc383631731)

[randFloat.test 352](#_Toc383631732)

[randInt.test 353](#_Toc383631733)

[randProperty.test 354](#_Toc383631734)

[scaleCoord.test 354](#_Toc383631735)

[validateDirection.test 355](#_Toc383631736)

[Worker.depositingFood.test 356](#_Toc383631737)

[Worker.draw.test 358](#_Toc383631738)

[Worker.findingFood.test 360](#_Toc383631739)

[Worker.gettingFood.test 362](#_Toc383631740)

[Equal 365](#_Toc383631741)

[Test 368](#_Toc383631742)

[testCase 371](#_Toc383631743)

[Sources 375](#_Toc383631744)

[Ants and their behaviour 375](#_Toc383631745)

# Section 1: Analysis

## The problem: identification and background

### Problem identification

The problem addressed in this project is to develop a computer-based tool which will enable secondary school teachers of Biology at a school in Bath to illustrate how evolution occurs in a population. This problem was identified from interviews with Biology teachers which found that the subject of evolution is complex and difficult to teach to younger pupils aged around 12-13.

These interviews found that pupils in this age range find it difficult to grasp the abstract ideas of evolution as this involves following an argument with several stages to it and a number of novel concepts such as ‘genes’, ‘mutation’, ‘selection’ and ‘inheritance’. The interviews also found that presentations and handouts failed to capture the dynamic nature of evolutionary change and ‘bring it to life’. Further, due to a shortage of suitable tools and materials it is difficult for teachers to set meaningful classroom exercises on evolution and to encourage pupils to do independent learning outside of the classroom.

As a result of these problems in teaching the topic teachers have found that some pupils can have common misconceptions about natural selection. For example, some pupils persist in believing in Lamarckism - the idea that an organism can pass on characteristics that it acquired during its lifetime to its offspring. It was felt, in the interviews, that if a computer-based tool was available it could illustrate the process of evolution, enable pupils to have more of a ‘hands-on experience’ and could be an excellent way for pupils to understand for themselves why ideas such as Lamarckism are wrong - rather than being taught simply that they are wrong.

Online research and face-to-face interviews with teachers (see Appendix A for details) have shown that there are several types of videos, handouts, presentations and other written materials currently available for school teachers on the subject of evolution. In addition, there are some computer-based tools which simulate how evolution occurs and which can, for example, allow different timescales and types of populations to be selected and different simulations to be run. These computer-based tools are evaluated and a new solution is developed in this project.

The aim of this project is to create a tool to be used by in Biology classes for years 9 to 11 in Kingswood School to demonstrate to pupils the fundamental concepts of evolution in a fun and interesting way.

### Problem background - teaching context

The teachers who were initially interviewed teach Biology in Kingswood and each have two classes a year of 15 – 20 mixed ability twelve to thirteen year old pupils. Evolution is taught as part of a Combined Sciences course as part of an introduction to the natural world and a 40 minute lesson is devoted to it. In these lessons the teachers cover the following:

* Some context – the ideas of Charles Darwin.
* A definition of evolution.
* An overview of how it occurs – introducing a number of ideas including the existence of differences within a population, members of a population being more/less suited to their environment, natural selection and the lengthy timescales over which evolution generally occurs.
* Examples are introduced – these can include more material on Darwin, on his travels and the Galapagos Islands, and well-known examples of evolution such as Darwin’s finches or the evolution of butterflies in contemporary Britain and their response to pollution.
* What evolution is not/common misconceptions – Lamarckism.
* Lessons are a mixture of explanation by the teacher, using video or written materials to illustrate the subject, classroom discussion and taking questions.

Computers with internet access are available during lessons with one computer for every pupil.

## Description of available tools

The Biology teacher who has been interviewed is not currently using any computer-based simulation tools as part of their lessons on evolution. However, some tools do exist and this section assesses the leading examples.

The tools assessed fall into two categories: a) tools focused specifically on evolution, and b) more broadly-based tools - for example considering the impact of disasters on animal populations. The table below summarises the tools which have been assessed.

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Assessment** |
| **“Who Wants to Live a Million Years?”** | * Aimed at 12 - 14 age group * Focused on evolution * Learning and simulation sections, together with a quiz, glossary and Darwin biography * Shows impact of a change in environment on a population and effects of natural selection | * Learning and simulation sections * Poor treatment of mutations * Poor representation of gradual evolution processes * Not very engaging/fun * Minimal user interaction * Overall – too high level |
| **“Stop Disasters!”** | * Aimed at 16+ age group * Produced by the UN * Shows the impact of disasters | * Engaging and sophisticated * Good level of user interaction * Good information and links * Focused on disaster recovery |
| **“Cambrian EXPLOSION”** | * Aimed at 16+ age group * Focused on the Cambrian explosion, a period of time when evolution accelerated creating a large diversity of new species in a short period of time * Shows the evolution of locomotion in a variety of creatures using a genetic algorithm | * Good treatment of mutation and concept of evolution. * Good variety of settings which can be changed to focus simulation * Good use of statistics/data presenting them to users as an easy to read graph * Simple and easy to use * Little documentation of complex settings * Good use of simulation output (visual representation of evolution) * Too advanced for 12 – 14 year olds * Focused on development of locomotion not evolution more generally |

The tool “Who Wants to Live a Million Years?” (<http://science.discovery.com/games-and-interactives/charles-darwin-game.htm>) is an example of a free, independent-learning tool designed to help teach the concept of evolution. It is a flash-based web application aimed at teaching the basic concepts of evolution by demonstration. The content and graphics indicate that it is aimed at pupils aged 12 – 14.

The tool contains a learning section (‘Learn about Evolution’) and a simulation section (‘Play the Survival Game’). The learning section shows the user both text and pictures in a fun, animated way to help engage users. The simulation section proceeds as follows:

* It asks the user to choose a starting population and select variations within the population.
* It then demonstrates the impact of several changes in the environment (e.g. a warmer climate, a new predator or a natural disaster) on the numbers of the population - to show how only those best suited to the new environment will survive and reproduce while the others will not.
* If a population is wiped out the user is asked if they want to return to the beginning and select a different set of characteristics to test whether they will improve survivability.
* The simulation takes about 2 minutes to complete each time, with users expected to run several simulations.

A number of criticisms of this tool can be made:

* The simulation gives a misleading representation of natural selection as it presents mutations as non-random events (selected by the user) while in reality they are totally random and cannot be chosen.
* It is not clear what the different mutations are - different versions of an animal are shown in crude terms (e.g. one has longer fur, another has a longer body).
* The results of a change in environment are simply presented – there is no explanation why the different versions of a population are increasing or decreasing in number.
* The simulation is not very engaging: there is minimal user interaction (which does not make it fun) with just a few clicks from start to finish.
* Overall, the tool is aimed at young children to provide some ‘fun’ but it provides only a very crude view of evolution.

Another example of a flash-based web application is “Stop Disasters!” ([http://www.stopdisastersgame.org](http://www.stopdisastersgame.org/)). The focus of “Stop Disasters!” is not evolution but preventing and responding to disasters such as tsunamis – it is produced by the United Nations and the International Strategy for Disaster Reduction. It is aimed at older children than “Who Wants to Live a Million Years” and has a game section, a section showing high scores from the game, information on the largest disasters in recent history and teacher resources about disasters.

The simulation tool can be used to teach students about disaster prevention, as part of Geography or other lessons. The objective of the game is to assess the risk of a disaster and to reduce the damage to the human population when one does occur. The simulation section proceeds as follows:

* The user selects a scenario e.g. ‘Coastal Village – tsunami’, and difficulty level (which is generated by the level of detail of the maps used).
* A disaster occurs and the user is given a mission e.g. ‘construct accommodation for 400 people with a budget of $35,000’.
* The user then spends the budget to purchase various materials.
* A score is shown e.g. number of population successfully housed.
* The simulation takes about 10 minutes to run, depending on the choices made by the user.

“Stop Disasters!” is much more sophisticated and interesting than “Who Wants to Live a Million Years?” as it allows the user to interact with it more fully, it has better graphics and feels more like a game with choices – rather than a few simple steps over which the user has little influence. It does not deal with evolution but is a good example of a sophisticated interactive learning tool.

*Note*: Although “Stop Disasters!” is not specifically relevant to Biology it is useful to evaluate it as it shows how a simulation can be used as part of the teaching process.

A web based application using JavaScript is “Cambrian EXPLOSION” (<http://www.cambrianexplosion.com/>). The application is focused on showing the evolution of locomotion in creatures. It uses a generic algorithm to generate creatures with different mutations and then runs a simulation measuring how far they move in a specific amount of time. From this it calculates the fitness of each generation and draws graphs of how the fitness varies with each generation - thus showing the concept of evolution (small positive mutations in each generation add up as that species has a greater change of reproducing thus transferring the mutation onto the next generation). This tool is quite sophisticated and complex; although its graphics are not as good as “Stop Disasters!” it makes up for this in its simulation customizability.

In summary, the current tools which have been analysed have a number of advantages and disadvantages and none have been judged to meet all the relevant criteria. While “Who Wants to Live a Million Years?” is focused on evolution it has a number of drawbacks in the user experience and in the level of detail it shows about evolutionary processes. “Stop Disasters!” has a better user experience, being fun and informative and with significant user interaction, but is not focused on evolution. “Cambrian EXPLOSION” is a good simulation of evolution in species however it may be too advanced for the 12 - 14 year olds and also focuses only on the evolution of locomotion and not the general evolution of animals.

## Identification of prospective users

Two types of users of the system have been identified – teachers in the classroom and pupils. Teacher users are teachers of Biology in Kingswood who require an introduction to evolution for classes of 10 - 15 pupils aged 12 - 14. Teacher users have an adequate knowledge of computing and applications. Teachers will use the tool to demonstrate the basic concepts of evolution to the pupils, and will act as an administrator for the tool i.e. deciding when it is to be used and setting it for homework.

Pupil users are aged 12 - 14, are of mixed ability and have already had some background in science. They lack any detailed knowledge of evolution, although they are likely to have heard of it, and may find the idea challenging. Pupil users know how to follow simple instructions on a computer and use simple applications. Pupils will use the tool to see how the concepts of evolution taught to them in the classroom can be illustrated and ‘brought to life’.

## Identification of user needs and acceptable limitations

Analysis of user needs and acceptable limitations has been undertaken for both sets of users.

### User needs – teachers

Teachers require a tool that quickly and easily illustrates the concept of evolution in a manner that is understandable and interesting to children and which is easy to administer. Teachers’ more detailed requirements are:

1. The tool should not be over-complex in its ideas, as this would defeat the point of simplifying the teaching of the concept.
2. As the tool is to be used as both a teaching tool and a tool to prompt independent learning it must be able to be used within the classroom (or computer room) as well as by the pupils in their own time or for homework at home.
3. The tool needs to be able to be used by 12 - 14 year old pupils, either individually or in small groups of 4 - 6, and must be simple to use with minimum instruction required – no more than 5 minutes.
4. Setting up and running a simulation should take no more than 5 minutes per simulation by pupils.
5. Total time for using the tool – from instruction to the analysis of results – should be about 22 minutes.
6. The tool, together with any linked, content must be fun and engaging for this age group.
7. All pupils in the classroom and outside the classroom should have equal access to the tool and there should be no minimum system requirements (e.g. owning a smart phone), so that everyone can use it.
8. When first introduced, the tool must be demonstrated by a teacher through a projector so that the pupils can use the tool correctly and use all features to their maximum effectiveness.

### User needs – pupils

1. The tool must be suitable for 12 - 14 year olds in terms of the complexity of content and the ease of use.
2. It must be fun and engaging –pupils should enjoy each simulation and want to run several.
3. It must be capable of communicating clear learnings.
4. It must be capable of being accessed by all pupils both inside and outside the classroom.

### Acceptable limitations - software

1. The tool must work on the major web browsers:
   1. Internet Explorer 9 and upwards (current school version).
   2. Firefox version 20 and upwards.
   3. Chrome version 22 and upwards.
2. The tool must be able to run on Windows 7 which is the current school operating system.

### Acceptable limitations - hardware

1. The tool must run smoothly on the computers provided in the ICT rooms.
   1. Most computers are fitted with: Windows 7; Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz; 2.00 GB Memory. Note: Smoothly means with no noticeable judder (≥15fps) with standard use of the tool.
2. Only a mouse and keyboards can be used as input devices for the system as these are all that are provided by every work station.
3. The tool should be able to scale up to resolutions larger than 800 x 600 so that it can be displayed through a school projector onto an interactive whiteboard.

### Acceptable limitations - simulation

1. The tool should not be expected to be completely accurate and approximations may be made to avoid increasing the complexity of the simulation beyond the scope of the teaching.
2. The simulation should not be expected to model realistic ant movements and only model a 2D representation of ant movements.
3. The simulation should not be expected to model a realistic environment.
   1. However it should simulate basic regenerating food sources.
   2. It should only simulate a 2D terrain.
4. The simulation should not simulate the complete lifecycle of ants i.e. it should only model adult ants.
5. The simulation should not model the inner workings of the nest only the concept of the nest -where the Queen lives and new ants are born and where food is stored.

## Data sources and destinations

The pupils will input values through buttons and sliders into the simulation via the user interface. The values selected will be used to change how the simulation runs and will therefore require heavy processing (running through the simulation).The simulation will output the result in an animation onto the screen.

## Data volumes

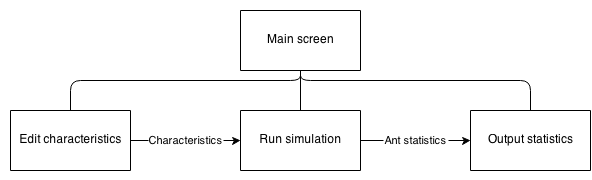
No data is saved as all data is processed in real time and outputted data is shown to the user immediately. So the size of the program is the only data volume, this will be less than 1mb.

## Analysis data dictionary

* Characteristic – a variable feature of an ant which changes it in some way e.g. how fast it can move.
* Species – a collection of unique characteristics define a new species in the simulation.
* Mutation – a random event which alters a characteristic.
* Ant nest (or just nest) – the home of the Queen ant and where it reproduces as well as a general food store.
* Queen ant – the head of the species and the only ant capable of reproducing.
* Worker ant – an ant whose purpose is to collect food and supply it to the nest.
* Soldier ant – an ant whose purpose is to guard the nest and other ants from attack, and also to attack ants of a different species.
* Pheromones – chemicals deposited by ants when they move which can be detected by antennae and used to alert other ants of danger or as a trail to food. Pheromone concentration slowly degrades over time due to evaporation.
* Trial pheromone – deposited by worker ants and used as trails to and from food.
* Fitness – the ability to survive and reproduce
* Food – a source of nutrition which is in constant demand by ants. Without a constant supply of food an ant will die. Food will be needed by the nest to create new ants.

## Structure and data flow diagrams

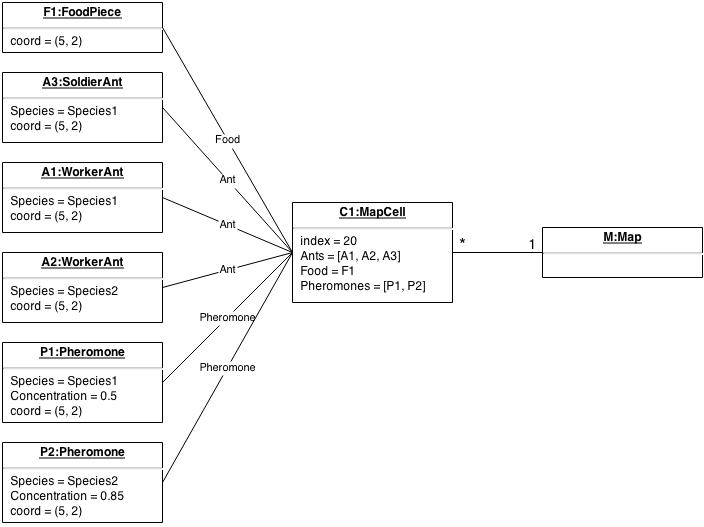
### Structure diagram



## Entity-relationship (E-R) model

This is not applicable as no database manipulation is required.

## Object analysis diagram



## Objectives for the proposed system

### Ant simulation objectives

1. A tool which **illustrates the concept of evolution**. This concept is summarised in Appendix B. The main ideas of the concept of evolution should be covered:
   1. **Organisms have characteristics** which determine their species.
   2. As a result of the mutation of characteristics, **variations capable of being inherited** exist within populations of organisms.
   3. Organisms **produce more offspring than can survive**.
   4. These **offspring**, with their different inheritances, **vary in their ability to survive and reproduce**.
   5. In conditions with competition between organisms for survival and reproduction those **organisms with traits that give them an advantage over their competitors pass these advantageous traits on**, while traits that do not confer an advantage are not passed on to the next generation. As a result we have the ‘survival of the fittest’ and a gradual change in populations – they change or may even die out.
2. The simulation should be able to generate **random food placement**.
3. The simulation should be able to simulate basic **growth of food**.
4. The tool should provide a **simulation involving movement** since evolution involves dynamic change and a static tool is not appropriate.
5. The simulation must simulate the three basic types of ants:
   1. The **worker ant** – locates and collects food.
   2. The **soldier ant** – guards and attacks ant species.
   3. The **queen ant** – creates a nest.
6. The simulation must **model a basic nest** capable of:
   1. Producing new ants.
   2. Being attacked and eventually destroyed.
7. The ants should have variable basic characteristics Including:
   1. **Speed** the ants move at.
   2. The **reproduction rate.**
   3. The **amount of food ants are able to carry.**
   4. The ants’ **eyesight** i.e. the distance an ant can see.
   5. The ants’ **antenna size** i.e. the distance the ant can smell pheromones from.
8. The simulation should be able to **introduce random mutations** (so pupils can see how the ants most suited to the environment will survive).
9. The simulation must be able to **show multiple different species at the same time**, so that their fitness can be compared by the pupils.
10. The simulation should model **energy intake from food** needed for an ant to survive depending on the ant’s characteristics. For examples ants with more favourable characteristics such as moving faster should require much more food than ants which have less favourable characteristics.
11. The simulation must be able to model **ants fighting**.
12. The simulation must model **pheromone trails** including:
    1. Their **creation** when an ant is moving.
    2. Their **evaporation** due to conditions.
    3. **How ants respond** to the trails:
       1. Following to find food.
       2. Following back to nest.

### Simulation interface features objectives

1. **Pause and play buttons** to pause and play simulation.
2. **Navigation buttons** to move around the simulation.
3. **Zooming** in and out of the simulation.
4. For each species (displayed when selecting nest) statistics measuring:
   1. The **amount of food stored in the nest** of a specific species.
   2. **Number of ants** in species.
5. **Editable characteristics** e.g. by sliders or buttons so that the user can change the simulation in real time.
6. **Selection of different species** so that their characteristics can be compared.

## Appraisal of potential solutions

### Smart phone app

A smart phone app written in Java (Android) or Objective-C (IOS) would run the simulation on a phone. This would fulfil the majority of the requirements:

* Due to smart phone portability the app could be used both inside and outside the classroom.
* Phones are internet-enabled and so the app can be downloaded from within the class room.
* The age group are familiar with the concept of apps and would be familiar with the interface.
* Smart phones are powerful enough to run a complex biological simulation and could maintain simulation accuracy.

However, there are some disadvantages from a smart phone solution. Not all pupils would have a smart phone and as a result not everyone in the class would be able to use the app. Furthermore, due to the fragmentation of smart phone operating systems the same app would need to be written for multiple different architectures (IOS, Android …) - possibly making the project unviable. Lastly, if the exercise is done in class there would be no way to know if the pupils where using the tool or using their phones for another use.

### Web application

A web application written in HTML5 and JavaScript is another potential solution. Using the web has a number of advantages.

* The internet is available on school and home computers.
* Access to the tool would not be a problem as almost everyone has access to a computer (i.e. in the school computer lab) thus fulfilling the users’ requirements.
* Even young pupils are very familiar with web application interfaces and as such teachers would only be required to provide a short introduction and pupils could quickly learn how to use the application.
* There would be little issue with cross compatibility as both html and JavaScript are web standards and implemented in all modern browsers including mobile browsers.
* It would be very easy to set up as it is essentially a webpage and therefore there is no need to download or install any applications – all that would be required is to visit the webpage.

However the complexity of the project may increase due to the use of multiple languages (HTML, CSS and JavaScript) and the added complication of configuring and maintaining a web server and domain. Furthermore a significant amount of time would have to be spent optimising the application to speed it up as JavaScript is much slower since it runs through a browser compared with natively written programs.

### Desktop application

A desktop application written in C/C++ or Java would have a number of advantages:

* The simulation could easily run due to C/C++’s speed and also access to a desktop.
* Like the web application in (HTML5, JavaScript) it could be used by all of the pupils at home or in the classroom fulfilling the objectives, as C/C++ can be compiled to run on multiple different operating systems.

However, the application would have to be cross compatible between the major operating systems (Windows, Mac and Linux) in order to make sure everyone could use it on their machine, thereby increasing the complexity of the program. Furthermore, the increased set up time of the tool would make it less user friendly (i.e. user must download and install the program).

### Pen and paper simulation

A non-technical solution to the problem would be to create a set of rules and then use counters on a gridded board to simulate how the ant’s behaviour would change depending on their input values. Random mutations could be introduced by using dice:

* The simulation could be used by all of the pupils.
* The simulation does not need computers and therefore the students do not have to change room to one equipped with computers, increasing the amount of time available.

However, a pen and paper simulation would be far too slow and the pupils may not be able to see the overall patterns emerging from it and therefore not learn the concepts of evolution. It would require more than one pupil to be fun and engaging and so cannot be done at home for homework. It would be tedious to record all of the statistics (a computer based solution could do this automatically). The scale of the simulation would be dramatically reduced to fit into the time allowed and would be less impressive as a result, making the simulation less interesting to pupils.

## Proposed solution

### Simulation

An ant simulation which models three basic types of ants:

* **Worker ants** – responsible for searching, collecting and depositing food.
* **Queen ants** – responsible for locating a new nest location.
* **Soldier ants** – responsible for defending and attacking other species.

It will also model a simple nest which has the ability to create new ants in a probabilistic manner.

Food must also be modelled. Food will be collected and eaten by ants to keep them alive. Nests will use food to create new nests i.e. a nest will require a certain amount of food to create a new ant.

Pheromone trails will also be modelled. These will be deposited by ants when they are returning to the nest with food. Pheromones will influence the direction of other worker ants of the same species.

It will be implemented as a single page web application with the simulation running as a local JavaScript script. Because of this the running speed of the script will be dependent on the performance of the computer running it. From analysis of the computers available to pupils in the Biology class rooms, the simulation should be able to smoothly run from all available computers if implemented in a modern web browser. A modern web browser such as Mozilla Firefox or Google Chrome will have to be used due to their superior JavaScript engines (Firefox uses SpiderMonkey and Chrome uses v8), which will allow the application to be run even on low end computers.

A web application and JavaScript have been chosen as there is no required installation, as there is with other solutions such as a desktop application in C/C++ or Java or even a mobile app. This will make the application easier and painless to use as it will only require the pupil to go to a website. Furthermore this will allow the application to be set up and operational faster than other applications. This will mean that more of the lesson will be spent on teaching rather than loading.

### Interface

The interface will be a single page web application designed in HTML and formatted in CSS. To display the simulation the HTML5 canvas will be used (introduced in the HTML5 specification). The application will be coded in JavaScript. This will not only allow manipulation of the canvas element but also allow the code to edit items on the page. This means data and information can be put onto the page with relative ease compared with other web scripting languages.

A site designed and styled in HTML and CSS will be very familiar with the pupils as the majority of websites on the internet are created this way. The site will use web standards set out by w3 (World Wide Web consortium - the main international standards organization for the World Wide Web) and common design practices to make a user-friendly website. This will allow pupils to quickly learn how the tool works - as it will be similar to other applications they have used on the web. (This is not as easy with a desktop application as there are many different GUI libraries (Qt, Tinkter, WxWidgets to name a few) used and no single standard, so a desktop application would be less intuitive to the user as it is less likely they have seen something similar.)

Another advantage of creating a web application is that it is cross platform. This cannot be said about desktop applications or mobile apps, as both generally require a substantial amount of effort to port to anther system. A website can be used on both desktop computers and smartphones. This leaves open the possibility of using phones or tablets in the classrooms in the future to run the simulation. Again, the web’s cross platform nature will allow the application to be easily used wherever the pupils want to use it, whether at school or at home. This cannot be said about desktop applications as the pupils may use a different operating system such as OS X at home while windows 7 at school.

## Justification of chosen solution

The chosen solution is a website based tool written in HTML5, CSS and JavaScript and using a school webserver backend to serve the website. This was chosen as the solution because it allows for all of the objectives to be achieved to their full with the least additional complexity.

* All simulation objectives can be accomplished with a well-written optimised program in JavaScript.
* The simulation can be easily displayed in an aesthetically pleasing manner with CSS to engage the pupils.
* The tool requires very little preparation time as it only requires the browser which is already installed on school computers.
* The tool can be easily used at home by pupils as it is a website and can be navigated to from anywhere with internet access and a web browser (which it is assumed pupils have at home).
* The tool can be easily displayed through a projector and should be able to scale to any sized monitor (as it is a website).
* The tool will have a shallow learning curve as pupils already know how to navigate web pages.
* The tool will be able to run on existing hardware and software.

# Section 2: Design

## Overall system design

### The simulation

The simulation will simulate the principles of survival of the fittest by using ants. It will simulate ant social structure as well as a basic environment for the ants to move in.

#### Evolution

Ants in the simulation will have a set of characteristics which define the ant’s behaviour. For example one characteristic of an ant is its eyesight; if a particular ant has very good eye sight, it can spot food from further away, thus being more able to survive. The characteristics implemented will be:

* **Speed –**the speed the ant can move.
* **Antenna size –** the size of the ant’s antenna (determines how far away the ant can smell pheromones).
* **Antenna angle –** the angle through which the ant can detect pheromones from its current direction.
* **Eye sight –** the range at which the ant can see objects.
* **Eye angle –** the angle through which the ant can see objects from its current direction.
* **Jaw size –** the size of the jaw of the ant (determines how much food an ant can carry).
* **Jaw strength –** the strength of the ant’s jaw (determines how much damage it can do when biting other ants).
* **Sting size –** the size of the ant’s string (determines how far the ant can attack).
* **Pheromone concentration –** the concentration of pheromones secreted by the ant.

A species is a collection of specific values of characteristics. There will be multiple species in the simulation. Ants will all belong to a species, and it is the species which determines the ant’s characteristics. When ants are born they will inherit the species which their nest has. However, when a queen is born there is a chance that the species the queen inherits is mutated. A mutation is the change of a single value of a characteristic in a species. If a queen inherits the mutated species, and if it goes onto creating a nest, then all ants born from this nest will inherit the mutation.

A mutation can either be beneficial or destructive e.g. better eyesight is beneficial as it allows ants to see further, while a decreased speed is destructive as it means that the ant will move slower. Ants which inherit mutations that are beneficial are more likely to be successful in surviving, thus they will reproduce more ants. This means that the species will be passed onto more generations. This is the basic process of evolution and how it will be implemented in this simulation.

To decrease the advantage of some mutations there is a cost associated with characteristics. If a characteristic is more beneficial it will cost the nest a lot more *energy* (health in the simulation) to create an ant.

#### Ant behaviour

All ants must eat a certain amount of food otherwise they will lose health. Ants will have different amounts of health depending on their characteristics, if health is below 0 the ant dies.

There will be three types of ant:

* **Worker ant** – responsible for finding, collecting and bringing food back to the nest.
* **Soldier ant** – responsible for attacking other ant colonies - destroying their nest or killing ants from the other nest to disrupt their collection of food, leading to the collapse of the nest because of starvation. Soldier ants are also responsible for defending ants from attack from other species and for defending the nest.
* **Queen ant** – creates a nest, and reproduces new ants.

**Trial pheromones** are continually laid down by worker ants; they will slowly diffuse into surrounding areas and will also slowly evaporate in the environment. When an ant finds food it will try to make its way back to the nest and, as it does so, will lay down pheromones. These pheromones will then be followed by other ants either to the nest or to the food. Ants which have followed the trail and found food will lay down their own pheromones. Each time more pheromones will be laid down making the path more concentrated. Ants will be more likely to follow very concentrated routes when looking for food. Soldier ants will act as sentries and stand along busy trails defending their worker ants.

### Interface

At the start of the simulation the user will adjust the characteristics of the first species or choose random characteristics. Once the simulation is running the user will be able to edit ants’ characteristics or leave the starting characteristics and watch how the simulation plays out.

During the simulation information about the species in the simulation is added to the page to allow the user to review the information and adjust characteristics or judge how well the species is doing. The information presented is:

* **Colour** – this will be useful as it will show the user the colour of the species so that they can be easily spotted on the simulation.
* **Number of ants** – the number of ants in the species. This is clearly useful as it shows the user how successful the species is.
* **Number of nests** – the number of nests in the species. This is useful as it gives a guide to how likely the species is to grow as more nests will generally mean more ants.
* **Total amount of food in nests** – this is the combined amount of food held by the nests in the species. This again is useful as it gives a guide to the rate at which the species can expand. More food will mean more ants and a more successful species.

The user will be able to adjust characteristics and see them update live onto the simulation (i.e. without a restart). This is a useful for the user as it allows them to explore how each characteristic affects ants in the simulation. It will be useful when the user first uses the simulation to quickly allow them to understand all of the characteristics. As the user adjusts the characteristics they will get instant feedback in the form of text which shows the species’ cost as well as the cost of each ant in terms of food. The user will also be able to do the control the characteristics as a whole:

* **Update** - push the modified characteristics to the simulation. This is needed rather than an instant update because it acts as a buffer. If a user accidently edits a characteristic it should not be pushed to the simulation.
* **Default values** - the update all characteristics to default values. This is needed if the user has changed characteristics and has not been able to create a viable ant. The default values will be designed to create a species which is initially successful - this is a useful demonstration to prospective users.
* **Random -** this will set all characteristics to random values. This is useful to the user to show how even extreme mutations can be successful.

The user will be able to zoom in and out of the simulation as well as pan around the entire simulation using mouse gestures e.g. dragging and scrolling. It will also be possible to control the simulation timescale. The user will be able to:

* **Run/Pause** - start and stop the simulation. This is clearly useful as it allows the user to observe the entire simulation in a paused state.
* **Step** - go a single frame into the future. This is useful if the user wants to view an event which happens quickly e.g. an ant being chased by a soldier ant.
* **Restart** - restart the simulation if it doesn’t go to their liking or if all ants die off.

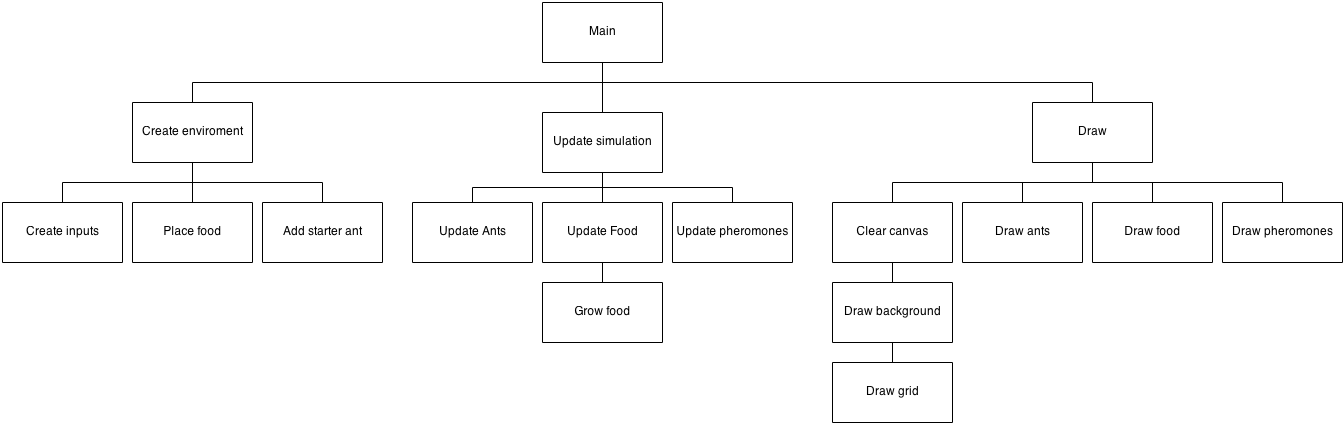
## Description of modular structure of system

The following is an analysis of entities which will be in the simulation, their expected behaviour, actions and properties.

*Note*: Classes defined may change due to unforeseen design decisions.

*Note*: **Number** type is any real number.

### Overall system



### Ant

The properties of a basic ant are the following:

* **Position** - the location of the ant.
* **Direction** - the direction the ant is facing.
* **Size** - the size of the ant.
* **Species** - the species the ant belongs to.
* **Nest** - the nest where the ant was born.
* **Health** - the current amount of health the ant has.
* **Health Threshold** - the level of health below which the ant is hungry.
* **Health rate** - the rate at which an ant will lose health.
* **Goal** - the current goal the ant is trying to complete e.g. find food.
* **Target** - the target the ant is heading to e.g. a piece of food.

An ant can also perform the following basic actions:

* **Take food - t**akes a single piece of food.
* **Get food - h**ead towards a piece of food.
* **Find food target - f**inds a target piece of food within viewing distance.
* **Scan - s**can surroundings for items of interest which can be seen e.g. other ants or food.
* **Smell - s**mell the surrounding area for pheromones.
* **Secrete - s**ecrete pheromones.
* **Walk - w**alk around either randomly or following pheromone trails.
* **Die.**

The following is the proposed class designed from the analysis of a basic ant.

|  |  |
| --- | --- |
| **Property** | **Purpose** |
| size [[number, number]] | The width and height of the ant |
| coord [[number, number]] | The x and y coordinate of the ant |
| direction [number] | The direction the ant is facing |
| id [integer] | The unique identifier of the ant, if it is required to be looked up |
| species [Species object] | The species the ant belongs to |
| type [integer] | The type of ant e.g. worker |
| nest [Nest object] | The home nest the ant was born from |
| colour [string] | The colour the ant will appear |
| health [number] | The amount of health the ant has |
| hungerThreshold [number] | The threshold bellow which the ant will be hungry |
| healthRate [number] | The rate at which the ants health decreases each tick |
| alive [boolean] | Determine if the ant is alive or not |
| goal [integer] | The current goal the ant is trying to complete e.g. get food |
| target [[number, number]] | The target coordinate of an item of interest e.g. a piece of food |
| itemsInView [Array] | An array of items in view i.e. food and other ants |
| pheromonesInRange [Array] | An array of pheromones in range |
| prioritizedDirection [number] | The direction the ant wants to move in |

|  |  |
| --- | --- |
| **Method** | **Purpose** |
| addToMap | Add the ant to the map |
| removeFromMap | Remove the ant from the map |
| isHungry | Determine if the ant is hungry or not |
| takeFood | Take a single piece of food from the cell the ant is standing on |
| atNest | Determine if the ant is standing on top of the nest. This will be used to determine when a worker ant can drop food or when a soldier is near the nest |
| seeNest | Determine if the ant can see the nest. This will be used by worker ants so they know which direction to head in to get to the nest. It will also be used by soldier ants when guarding the perimeter of the nest |
| findFoodTarget | Pick a piece of food to target |
| getFood | Move towards a piece of food and stand on top of it until it has been completely used |
| useFood | Decide how to use a piece of food e.g. eat the piece of food or carry it |
| scan | Scan visible surroundings for items of interest, use this information to populate itemsInView |
| smell | Similar to scan however only detects pheromones in range, populates pheromonesInRange |
| secrete | Secrete a single pheromone onto the cell the ant is currently standing on top of |
| wonder | Decide how to move the ant e.g. follow a pheromone trial or wander randomly |
| move | Move the ant a certain distance in the direction the ant is facing |
| die | Kill the ant and remove it from the simulation |



### Worker

A worker ant is a specific type of ant; its purpose is to find, collect and deposit food to the nest. The worker ant must be able to perform the following actions:

* **Determine if it can carry food** - does the ant have enough room to carry extra food.
* **Deposit food -** navigate to the nest and drop food at the nest.

The worker class:

*Note*: inherits from Ant.

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Carrying [integer] | The amount of food the ant is carrying |
| carringThreshold [integer] | The threshold above which the ant will return to the nest and deposit food, if it cannot see any more |

|  |  |  |
| --- | --- | --- |
| **Method** | **Parameters** | **Description** |
| canCarry | N/A | Determines if an ant can carry food or not |
| depositeFood | N/A | Navigates towards the nest, stands on top of it and drops food |
| dropFood | N/A | Drops a single piece of food |
| useFood | N/A | Determines the best use for food i.e. eat the food or carry the food |
| doTask | N/A | Performs the actions required to complete the current goal |
| updateGoal | N/A | Determines the current goal of the ant |
| updateHealth | N/A | Updates the ant’s health |
| draw | N/A | Draws the ant |
| update | N/A | Runs through a complete update of the ant. Also determine if ant has died or not |

### Soldier

The soldier ant is a type of ant; it is responsible for defending ants from its own species and attacking other species. The ant therefore is required to:

* **Determine if there are soldiers in view** - this is used so that the soldier ants can spread out i.e. change guarding position if they can see another ant so that they spread out.
* **Determine if there is food in the view** - this is used for when guarding food, so they know where to guard.
* **Pick a target** - pick an ant in view to attack.
* **Follow other ants** - follow the ants they are attacking.
* **Attack** - attack the ants they have targeted i.e. reduce the ants’ health.
* **Guard the nest** - guard the ants’ home nest.
* **Guard pheromone trails** - guard the pheromone trails. This is done as they are used by worker ants and therefore by guarding them they are guarding a large number of worker ants.
* **Guard Food** - as food is the primary resource in the simulation, guarding it is an important feature to help one species dominate.

The soldier class:

*Note*: inherits from Ant.

|  |  |
| --- | --- |
| **Properties** | **Description** |
| targetAnt [Ant object] | The ant the soldier is targeting |

|  |  |  |
| --- | --- | --- |
| **Method** | **Parameters** | **Description** |
| nearSoldiers | N/A | Determine if the ant is near another friendly soldier ant or not |
| pickTarget | N/A | Pick a target ant from the ants in current view |
| follow | N/A | Follow an ant which is in view |
| attak | N/A | Attack an ant i.e. reduce its health |
| updateHeatlh | N/A | Update the ant’s health |
| guardNest | N/A | Control ants’ logic when guarding the nest. Ant should stand still somewhere close to the nest which where it is not in view of other friendly soldier ants |
| guardPheromones | N/A | Controls ants’ logic when guarding pheromones. Ant should walk normally around, following friendly pheromone trails if it finds them |
| guardFood | N/A | Controls ants’ logic when guarding a food source. Ant should first walk around until it finds food, then ant should stop near food |
| doTask | N/A | Perform the actions required to complete the current goal. If the ant has the following goals, perform these actions:   * Guard Nest – guard nest * Guard pheromones – guard pheromones * Guard food – guard food * Attack – follow ant and attack if close |
| updateGoal | N/A | Determine the current goal of the ant:   * If no goal – Give ant a random goal * If guard nest – guard nest * If guard pheromones – guard pheromones * If guard food – guard food |
| draw | N/A | Draw the ant |
| update | N/A | Run through a complete update cycle of the ant. This includes:   * Updating the ant’s health * Scan and smell surroundings * Pick a target * Do task and update goal * Move ant and add it to the map |

### Queen

A queen ant is another specific type of ant. The queen ant is responsible for creating new nests and therefore needs to be able to:

* **Pick a nest site.**
* **Create a nest.**

The queen class:

*Note*: inherits from Ant.

|  |  |  |
| --- | --- | --- |
| **Procedure** | **Parameters** | **Description** |
| doTask | N/A | Decide what actions need to be done to achieve the current goal |
| updateGoal | N/A | Determine the current goal of the ant |
| pickDirection | N/A | Pick the direction the ant will go in, to make the nest |
| createNest | N/A | Create a new nest in the ant’s current location, and kill the queen ant |
| draw | N/A | Draw the ant |
| update | N/A | Run through the ants update cycle:   * Update ant’s health * Determine if has died or not * If haven’t reached nest then continue moving * Else, create nest and die |

### Species

A species is a collection of specific values of characteristics. A characteristic is a property of an ant e.g. an ant’s eyesight (how far the ant can see). The following basic characteristics will be implemented:

* **Speed –** the speed the ant can move.
* **Antenna size –** the size of the ant’s antenna (determines how far away the ant can smell pheromones).
* **Antenna angle –** the angle through which the ant can detect pheromones from its current direction.
* **Eye sight –** the range at which the ant can see objects.
* **Eye angle –** the angle through which the ant can see objects from its current direction.
* **Jaw size –** the size of the jaw of the ant (determines how much food an ant can carry).
* **Jaw strength –** the strength of the ant’s jaw (determines how much damage it can do when biting other ants).
* **Sting size –** the size of the ant’s string (determines how far the ant can attack).
* **Pheromone concentration –** the concentration of pheromones secreted by the ant.

There will be a chance that a species will be mutated; this will result in the change in the value of a characteristic. This can happen whenever an ant is created. However the mutation will only ever be passed on by queen ants, as normal ants cannot create new ants and therefore cannot pass on the mutation.

|  |  |
| --- | --- |
| **Properties** | **Description** |
| pheromoneConcentration [number] | The concentration of pheromones produced by the ant |
| reproductionRate [number] | The rate at which new ants are reproduced. Used by nest to determine when to create new ants |
| colour [string] | The colour of the ants and nests in the species |
| id [integer] | The unique identifier of the ant |
| ants [Array of Ant objects] | The array of ants belonging to the species |
| nests [Array of Nest objects] | The array of nests belonging to the species |
| chars.speed [number] | The value of the speed characteristic |
| chars.anteenaSize [speed] | The value of the antenna size characteristic |
| chars.jawStrength [number] | The value of the jaw strength characteristic |
| chars.jawSize [number] | The value of the jaw size characteristic |
| chars.stingSize [number] | The value of the sting size characteristic |
| chars.eyeSight [integer] | The value of the eye sight characteristic |
| chars.eyeAngle [number] | The value of the eye angle characteristic |
| chars.antennaAngle [number] | The value of the antenna angle characteristic |

### Nest

The nest implemented will be a very simple abstraction of actual ant nests. The nest must have the ability to recreate new ants. The nest must also be able to store/receive food from workers.

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Size [number, number] | The size of the nest in pixels |
| Coord [number, number] | The coordinate of the nest on the map |
| Id [integer] | The unique identifier of the nest |
| Species [Species object] | The species the nest belongs to, used when the nest creates new ants. These new ants will inherit the species of the nest |
| Health [number] | The health the nest has |
| hungerThreshold [number] | The threshold bellow which the ant is hungry and tries to save food e.g. does not create new ants below this level |
| healthRate [number] | The rate at which the ants’ health decreases |
| Alive [Boolean] | Determines if the ant is alive or not |

|  |  |  |
| --- | --- | --- |
| **Procedure** | **Parameters** | **Description** |
| reproduce | N/A | Determine which ant to create using probabilities |
| createAnt | Type [integer] – The type of ant which will be created | Create a new ant of a particular type; add it to the species ants list |
| die | N/A | Create a new nest in the ant’s current location, and kill the queen ant |
| draw | N/A | Draw the nest onto the screen |
| update | N/A | Run through the ants’ update cycle:   * Update the nests health * If the ant is still alive * If the nest will reproduce or not |

### Pheromone

A pheromone is a chemical signal deposited by ants to influence the direction of other ants i.e. lead other ants to food. A pheromone has the following properties:

* **Concentration –** the concentration of the pheromone.
* **Position –** the position of the pheromone.
* **Species –** the species of the ant which deposited the pheromone. This is important as in some species ants will only follow pheromones laid down by an ant of their own species.

Evaporation in of pheromones will also be implemented; this means that over time, pheromones will slowly reduce in concentration as some of the chemicals are evaporated away.

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Size [number, number] | The size of the pheromone in pixels |
| Coord [number, number] | The coordinate of the pheromone |
| Concentration [number] | The concentration of the pheromone |
| Species [Species object] | The species the pheromone belongs to i.e. the species of the ant which laid the pheromone |

|  |  |  |
| --- | --- | --- |
| **Procedure** | **Parameters** | **Description** |
| draw | N/A | Draw the pheromone onto the screen |
| update | N/A | Run through the ants’ update cycle:   * Reduce the concentration of the pheromone depending on the evaporation rate of the system. * If the concentration of the pheromone is below 0 remove the pheromone. |

### Food

A piece of food has the following properties:

* **Position** – the location of the piece of food.
* **Size** – the size of the piece of food.
* **Density** – the density/concentration of the food.

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Size [number, number] | The size of the piece of food in pixels |
| Coord [number, number] | The coordinate of the piece of food on the map |
| amount [number] | The amount/concentration the piece of food contains |

|  |  |  |
| --- | --- | --- |
| **Procedure** | **Parameters** | **Description** |
| draw | N/A | Draw the piece of food onto the screen |
| grow | N/A | Grow the piece of food by a certain amount depending on the environment. |

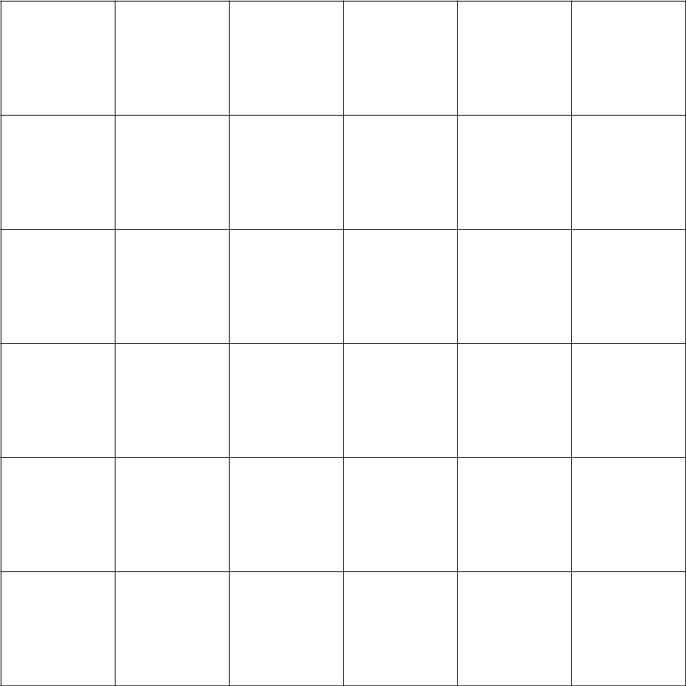
### Map

The map will model a 2d representation of a dirt based environment. Only the following entities will appear on the map:

* **Ants.**
* **Food.**
* **Pheromones.**
* **Nests.**

The map which will be implemented will be grid based. It will be made up by a fixed quantity of cells in a rectangular formation. Cells will be a fixed size. Ants, food, pheromones and nests will only be able to exist in a single cell i.e. one ant cannot be in two cells at the same time.

Height



Width

A Cell

Although the map will appear to be 2D and flat, in fact it will be a torus. This will simply mean that if an ant walks of the top of the map, they will appear to walk in from the bottom (also true for the left and right sides of the map). This is done so that the map will be able to appear larger is it is not limited by walls. It also more closely models the natural environment of ants i.e. not in a confined space.

### Configuration

The configuration options which will be available in the program include definitions of the environment in which the ants will be living in:

* **Food heath ratio** – the ratio of which one piece of food translates into so much health.
* **Food amount** – the amount of food which will be in the map.
* **Food grow amount** – the amount of food which grows each time.
* **Food grow rate** – the rate at which food grows.
* **Pheromone evaporation rate** – the rate at which pheromones evaporate in a single tick.
* **Max pheromone concentration** – the maximum concentration a pheromone can have.

Characteristics in the simulation will not only have a value but a number of other properties including:

* **Minimum value** – the minimum value the characteristics value can be.
* **Maximum value** – the maximum value the characteristics value can be.
* **Type** – this is the type which will be accepted as input i.e. an integer or a number or a string.
* **Description** – this is the description of the characteristic. This will be used for displaying the characteristic in the user interface.
* **Health modifier** – this is the amount at which a single increase in value will cost the species in terms of health.
* **Default value** – this will be used when the simulation is first loaded.

## Definition of data requirements (Design Data Dictionary)

| **Identifier** | **Description** |
| --- | --- |
| Characteristic | A collection of values relating to a property of an ant. This includes:   * **Value** – the value of the characteristic, type depending on the characteristic * **Maximum value** – the maximum value the value can have * **Minimum value** – the minimum value the value can have * **Type** – the type the value is e.g. integer or number * **Description** – a text description of the characteristic will be used in the user interface for hover text * **Health modifier** – how the characteristics value alters the species health cost * **Default value** – the default value of the characteristic, used when first opened |
| Species | A collection of specific values of characteristics |
| Mutation | An alteration in the value of a single characteristics value to a random value |
| Cell | A single square in a grid which can contain a single piece of food as well as many ants and pheromones |
| Map | A list of all cells in the map  *Note*: An index is the position of a specific cell in the map. Indices can be converted to and from coordinates using the algorithm stated in the algorithms section |
| Worker ant | The ant responsible for searching, collecting and depositing food back to the nest. This will be implemented as a class which inherits from the Ant class |
| Soldier ant | The ant responsible for attacking and defending ants. This will be implemented as a class which inherits from the Ant class |
| Queen ant | The ant responsible for founding a nest. This will be implemented as a class which inherits from the Ant class |
| Nest | Responsible for reproducing new ants and collecting food left by workers. This will be implemented as a class in the system |
| Food | An item on the map which is collected by ants and converted Into health to be used by ants to continue to live or by nests for creating new ants. This will be implemented as a class in the system |
| Pheromone | A chemical released by ants when they are finding their way back to their nest. This will be implemented as a class in the system |
| Goal | The aim of an ant. Different ants can have different goals depending on their purpose. The goals available will be:   * **None** – this goal will be used when ants are first created, they will be assigned a default task when they first run updateGoal in their update loop * **Find food** – this goal is used by all ants, it will cause the tasks needed to find food in the map * **Get food** – this goal is used by all ants, it will cause the tasks needed to collect a piece of food * **Drop food** – this goal is used by only worker ants, it will cause the tasks required to drop a single piece of food off at the nest * **Go to nest site** – this goal is used by the queen ant to head to the new nest site * **Create nest** – a goal used by the queen ant to cause the tasks required to create a new nest * **Guard nest** – a goal used by soldier ants to cause tasks for guarding the nest * **Guard pheromone** – used by the soldier ant to cause the tasks associated with guarding pheromone trails * **Guard food** – used by the soldier ants to cause task for finding and searching for a food source * **Attack** – a goal used by soldier ants to perform the actions required to attack another targeted ant |
| Task | A task is the specific action which will be required to be performed. Many tasks make up a single goal. An example of a task is picking up food, which will pick up a single piece of food |
| Target | A target can be a piece of food, an ant or a nest. The target will be used differently depending on the current goal. The target represents what the ant is heading towards |
| Direction | The direction the ant is currently facing |
| Prioritized direction | The direction the ant wants to move in |
| Size | The size of the object in pixels |
| Coord | A Cartesian coordinate |
| Health | Health will be a quantity used by ants to live. Ants start with a particular amount of health. This amount slowly decreases until at 0 the ant dies, the **healthRate** is the amount of health lost each tick. Health is restored by eating food. A single piece of food can be converted into a particular amount of health. The ratio of this is set by the **healthRatio** variable. The **hungerThreshold** is the amount of health bellow which the ant or nest is deemed to be hungry |
| Ant type | This is the type of ant e.g. a worker, soldier or queen |
| Id | A unique identifier, needed to compare if two objects are the same or not |
| Tick | A single loop of the entire program. Many ticks may occur each second |
| Species cost | The cost of the species i.e. the sum of all health modifiers applied to each characteristics value in the species |
| Food cost | The amount of food an ant is given when it is born. Part of this cost goes to the species cost, the left over is the amount of health the ant starts with. This will be different for each type of ant |
| Queen steps | The minimum and maximum number of ticks the queen will move for when locating a new nest |
| Worker, soldier, queen probability | The probabilities of each type of ant being created. The probabilities are relative to each other i.e. they will be normalised |
| Environment | The properties of the habitat the simulation is running in. Properties include:   * **Food heath ratio** – the ratio of which one piece of food translates into so much health * **Food amount** – the amount of food which will be in the map * **Food grow amount** – the amount of food which grows each time * **Food grow rate** – the rate at which food grows * **Pheromone evaporation rate** – the rate at which pheromones evaporate in a single tick * **Max pheromone concentration** – the maximum concentration a pheromone can have |
| wrapped coordinate | A coordinate which has been corrected for wrapping (due to the map being a torus) |
| map index | The index for a particular cell in the map |

## Description of record structure

This is not applicable as there are no databases used.

## File organisation and processing

This is not applicable as there is no data stored or retrieved from files.

## Validation required

Input validation will be done by sliders. This will reduce the amount of validation of input needed as it restricts the user to input values within a specific range and of a specific type. This is done to reduce the number of validation errors the user receives and thus streamlining their use of the application.

However the following characteristics will require extra validation and may produce validation errors or warnings:

* **Queen steps min and Queen steps max** – as these two characteristics act as a range of values it is important for min to the less then max. If this is not the case the user will be given an error and how to correct it.
* **Soldier, Queen and Soldier Food cost**– if an ant’s food cost is less or equal to the species cost the user should receive a warning as it means that the ant will be created with its health less than or equal to zero. This means that the ant will die instantly when it is created. This is likely to be not what the user wants to do and so a warning is issued (this is not an error however and the simulation can be run with ants with health less than zero).

The inputs provided to the system will be:

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Type** | **Range (inclusive)** | **Input via** |
| Speed | Number | 0 – 1 | slider |
| Antenna size | Integer | 0 – 10 | slider |
| antenna angle | Number | 0 - 2π (radians) | slider |
| eye sight | Integer | 0 – 10 | slider |
| eye angle | Number | 0 - 2π (radians) | slider |
| jaw size | Integer | 0 – 10 | slider |
| jaw strength | Integer | 0 – 10 | slider |
| sting size | Integer | 0 – 10 | slider |
| pheromone concentration | Number | 0 – 2 | slider |
| Food cost | Integer | 0 - | slider |
| Queen steps min and max | Integer | 0 - 2000 | slider |

## Identification of storage media

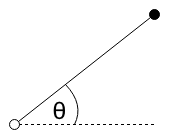
This is not applicable as there is no data stored.

## 

## Algorithms

### Angle between two points

The purpose is to find the angle from the horizontal in radians between two coordinates. This will be used by ants to target pieces of food or target other ants. It is needed so that the ants will know which direction to head in to get to a particular coordinate.



coord

target

**DECLARE** integer dx

**DECLARE** integer dy

**SET** dx = targetX - coordX

**SET** dy = targetY - coordY

**RETURN** atan2(dy, dx) + Math.PI / 2;

*Note*: atan2 is a built-in function to calculate the arctan corrected for the sector the angle lies in.

### Boundary

The map will act like a torus i.e. if an ant goes off one side of the map, it will reappear on the other side of the map. This pseudo code will update a coordinate to its *wrapped* coordinate if needed.

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Type** | **Description** |
| coordX, coordY | number | The coordinate of the entity |
| boundsXMin, boundsXMax | number | The minimum coordinate and maximum coordinate which the map covers in the X direction |
| boundsYMin, boundsYMax | number | The minimum coordinate and maximum coordinate which the map covers in the Y direction |

**IF** coordX < boundsXMmin **THEN**

**SET** coordX = boundsXMax - abs(coordX)

**ELSE IF** coordX >= boundsXMax **THEN**

**SET** coordX = coordX - boundsXMax

**ENDIF**

**IF** coordY < boundsYMin **THEN**

**SET** coordY = boundsYMax - abs(coordY)

**ELSE IF** coordY >= boundsYMax **THEN**

**SET** coordY = coordY – boundsYMax

**ENDIF**

**RETURN** [coordX, coordY]

*Note:* abs() is a built-in function which returns the absolute value of a number.

### Get a block of cells

It will be necessary to sample blocks of cells by ants to see what lies around them. This algorithm will return an array of all the cells which lie around the ant.

**DECLARE** block = []

**FOR** y = (coordY – sizeHeight) **to** (coordY + sizeHeight) **DO**

**FOR** x = (coordX – sizeWidth) to (coordX + sizeWidth) **DO**

block.push([coordX, coordY])

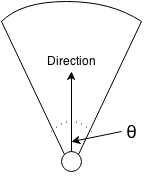
**ENDFOR**

**ENDFOR**

**RETURN** block

### Get a sector

Get sector will return the list of cells which lie within the sector at a particular coordinate facing a certain direction with an angle θ. This will be used by ants to sample the map in front of them to see any items of interests such as other ants, food and pheromones. Ants will have a certain viewing angle (the angle through which they can see).



**DECLARE** block = [];

**FOR** y = (coord.y – radius) **TO** (coord.y + radius) **DO**

**FOR** x = (coord.x – radius) **TO** (coord.x + radius) **DO**

**SET** searchCoord = {

x: x,

y: y

}

**SET** angle = angleTo(coord, searchCoord)

**SET** dist = distance(coord, searchCoord)

**SET** minSector = validateDirection(direction - angle / 2)

**SET** maxSector = validateDirection(direction + angle / 2)

**IF** angle >= minSector **AND** angle <= maxSector **AND** dist <= radius) **THEN**

block.push(searchCoord

**ELSE** **IF** (direction <= angle / 2 **OR** direction >= Math.PI \* 2 - angle / 2) **AND** (angle <= maxSector **OR** angle >= minSector) **AND** dist <= radius **THEN**

block.push(searchCoord)

**ENDIF**

**ENDFOR**

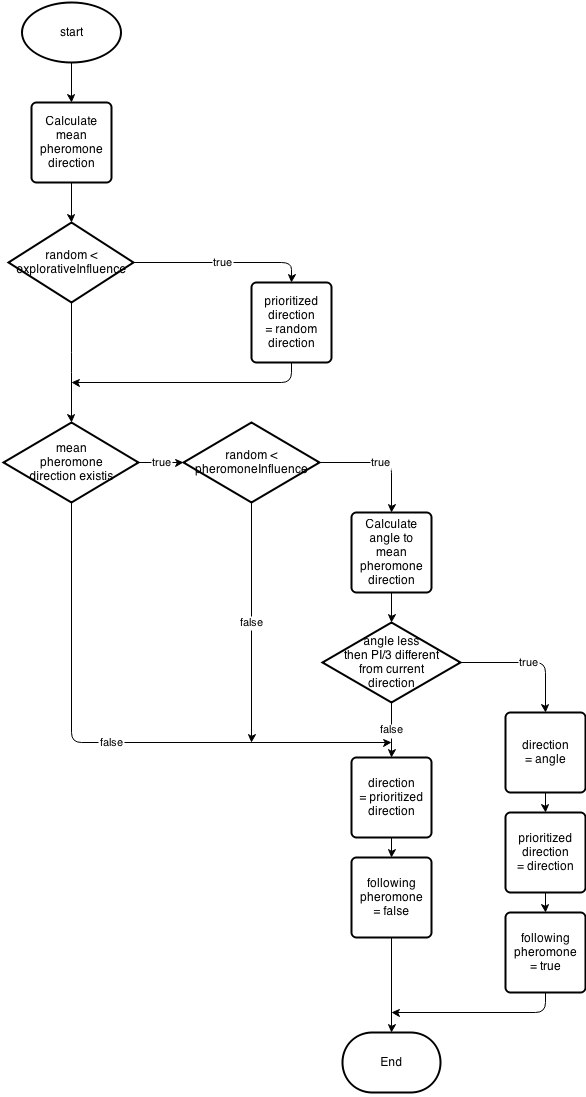
**ENDFOR**

**RETURN** block

### Ant non goal driven movement

A task which all ants will do is non goal driven movement. This means wandering around the map without a specific target. This is done when ants are searching for food, when soldiers are looking for their nest or guarding pheromone trails. Ants will walk randomly although their direction will be influenced by pheromones they come across. The higher the concentration of pheromones the more likely the ants will follow a particular direction.

The prioritised direction is used as a long term direction goal. This goal could last from 5 to 100 ticks. It is the direction the ant will tend towards and can change randomly. The explorativeInfluence is the chance that the prioritised direction will change. The purpose of the prioritised direction is to create straight paths (rather than ants constantly jerking around as they change direction randomly).



### Ant pheromone secretion

An algorithm is used to add a pheromone to the map. If there is already a pheromone of the same species on the map it will add to the concentration of pheromones - this means that pheromones will get stronger if ants secreting pheromones walk over them. This is a key feature of the simulation as it allows trails to form to lead to food.

**FOR** pheromone in cell **DO**

**IF** pheromoneSpecies == thisSpecies **THEN**

pheromoneConcentration += thisPheromoneConcentration

**IF** pheromoneConcentration > MAX\_PHEROMONE\_CONCENTRATION **THEN**

pheromoneConcentration = MAX\_PHEROMONE\_CONCENTRATION

**ENDIF**

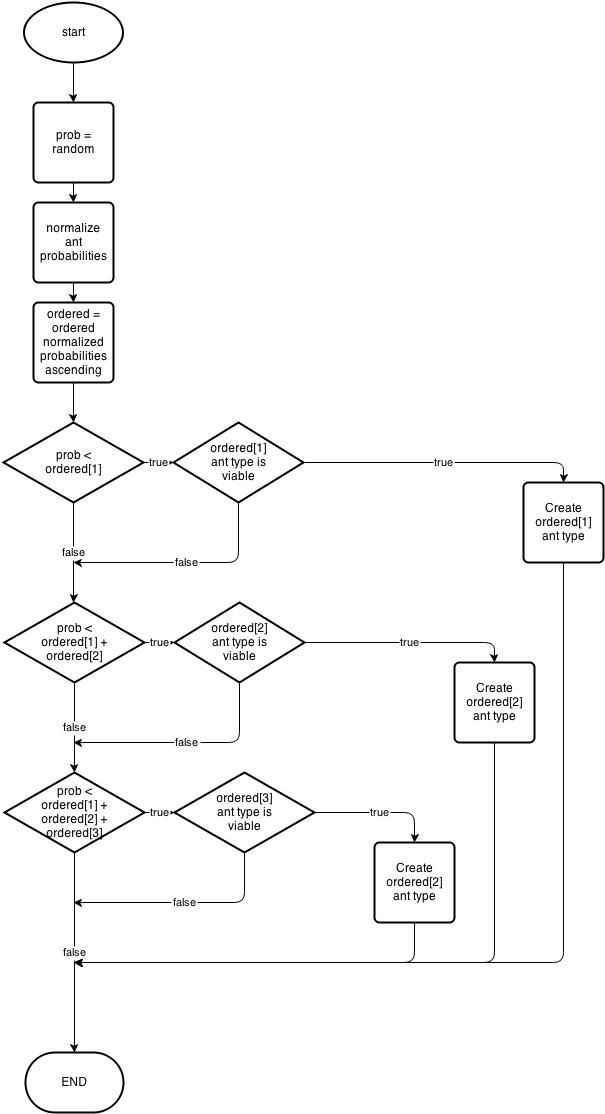
**ENDIF**

**ENDFOR**

pheromone = **new** Pheromone(this.species.chars.pheromoneConcentration, this.coord)

### Nest reproduction

An algorithm will be used by the nest to decide which ant will be created. This algorithm uses a probabilistic approach in that it is more likely that some ants will be created than others.

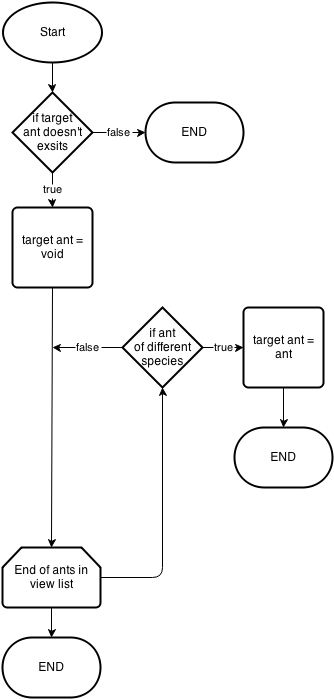


### WorkerWorker ant depositing food

### SoldierSoldier nest guarding

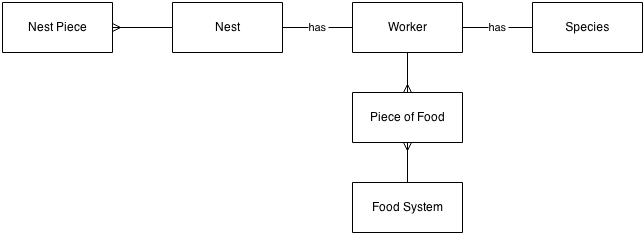
### Soldier pick target

This algorithm will be used by soldier ants to pick a target ant to attack.

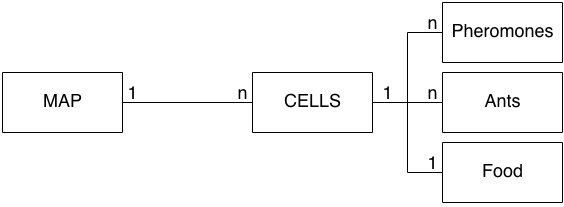


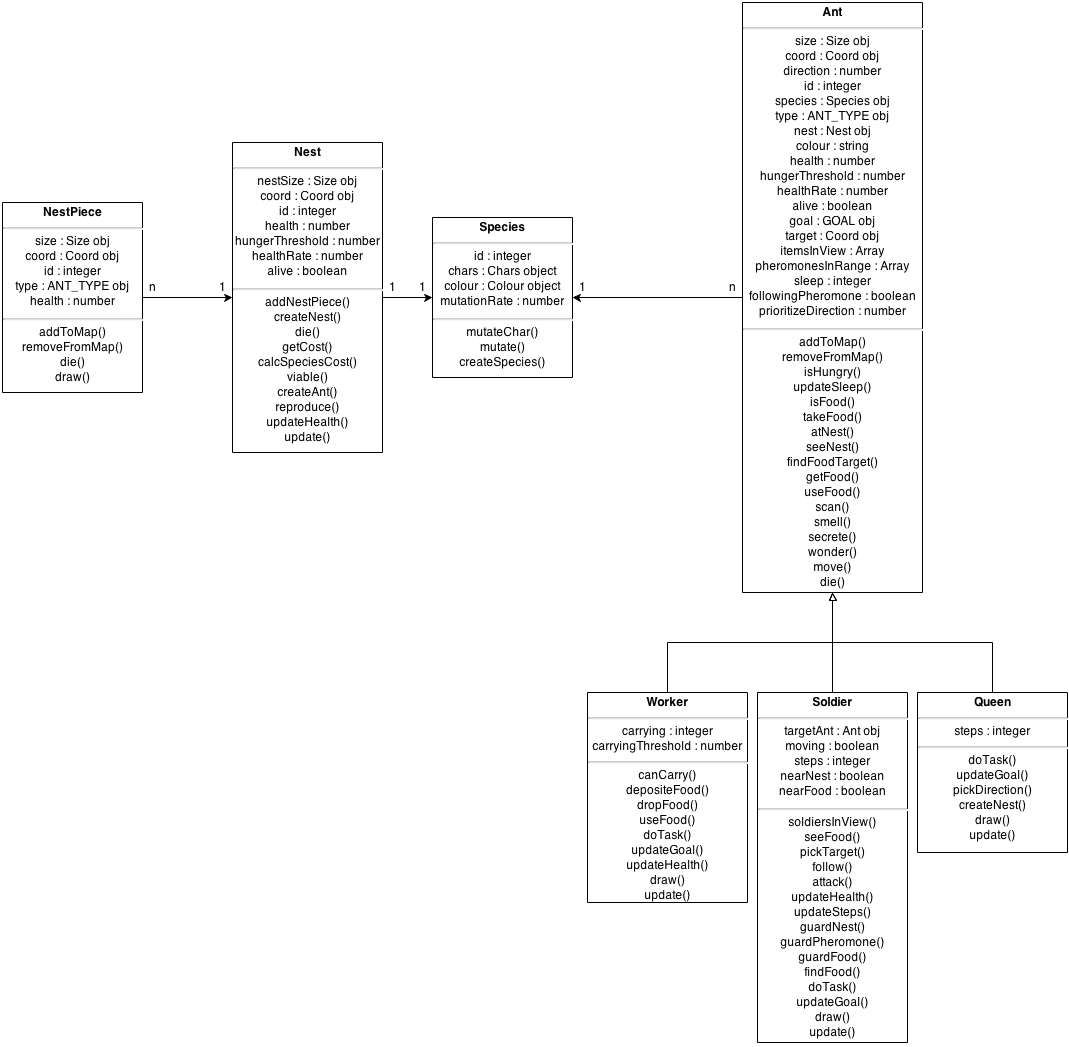
## Class definitions (diagrams) and details of object behaviours and entity relationship - antmethods

### Ant object behaviour

The diagram above shows the relationship between an ant and other objects in the simulation. This is true for all ants however the worker ant is slightly different. A worker ant can also hold multiple pieces of food.

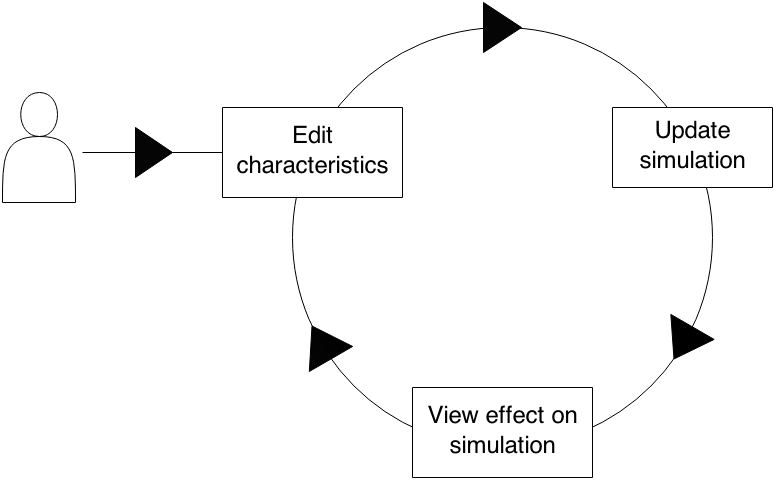
### Map object behaviour

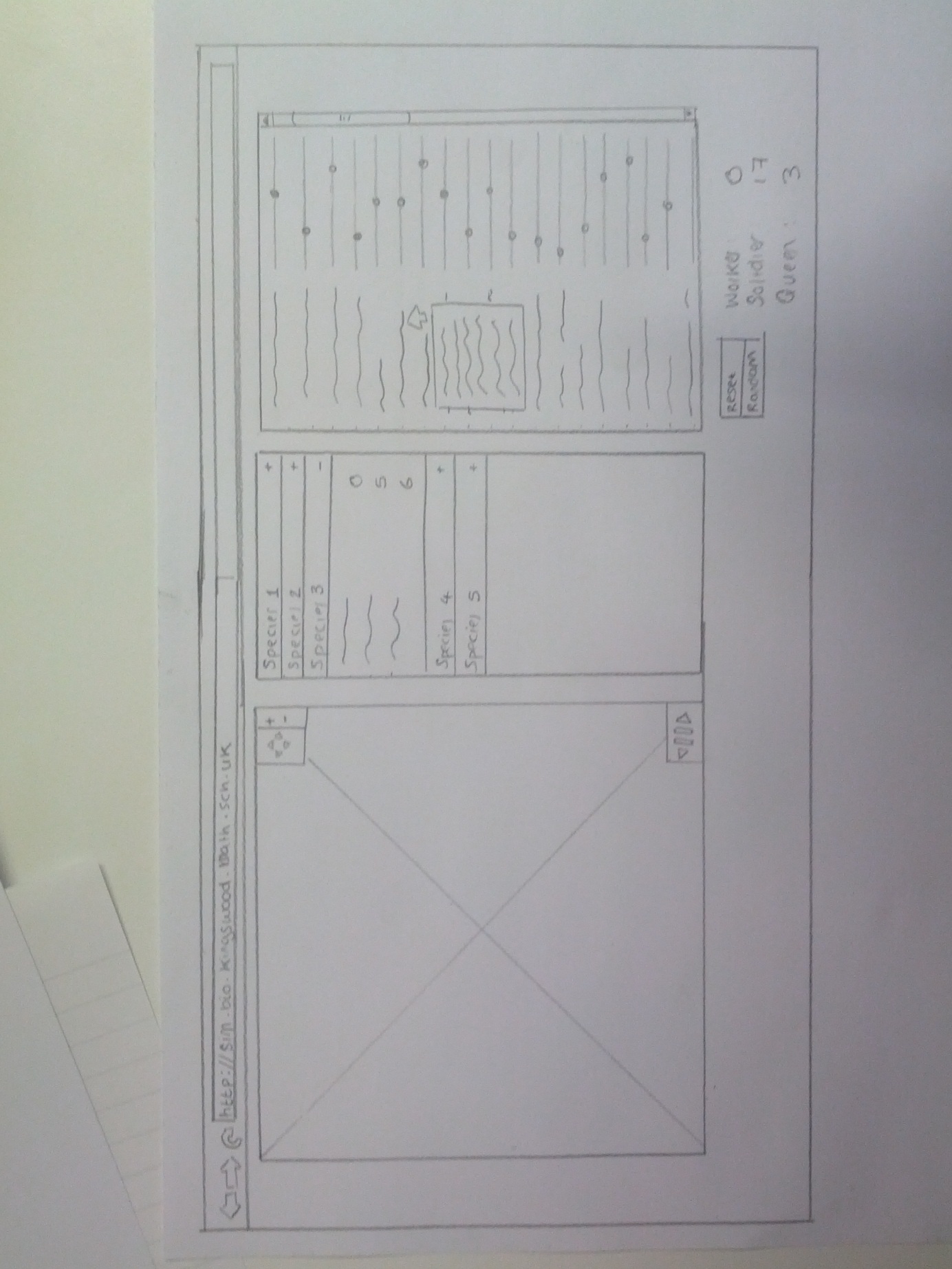


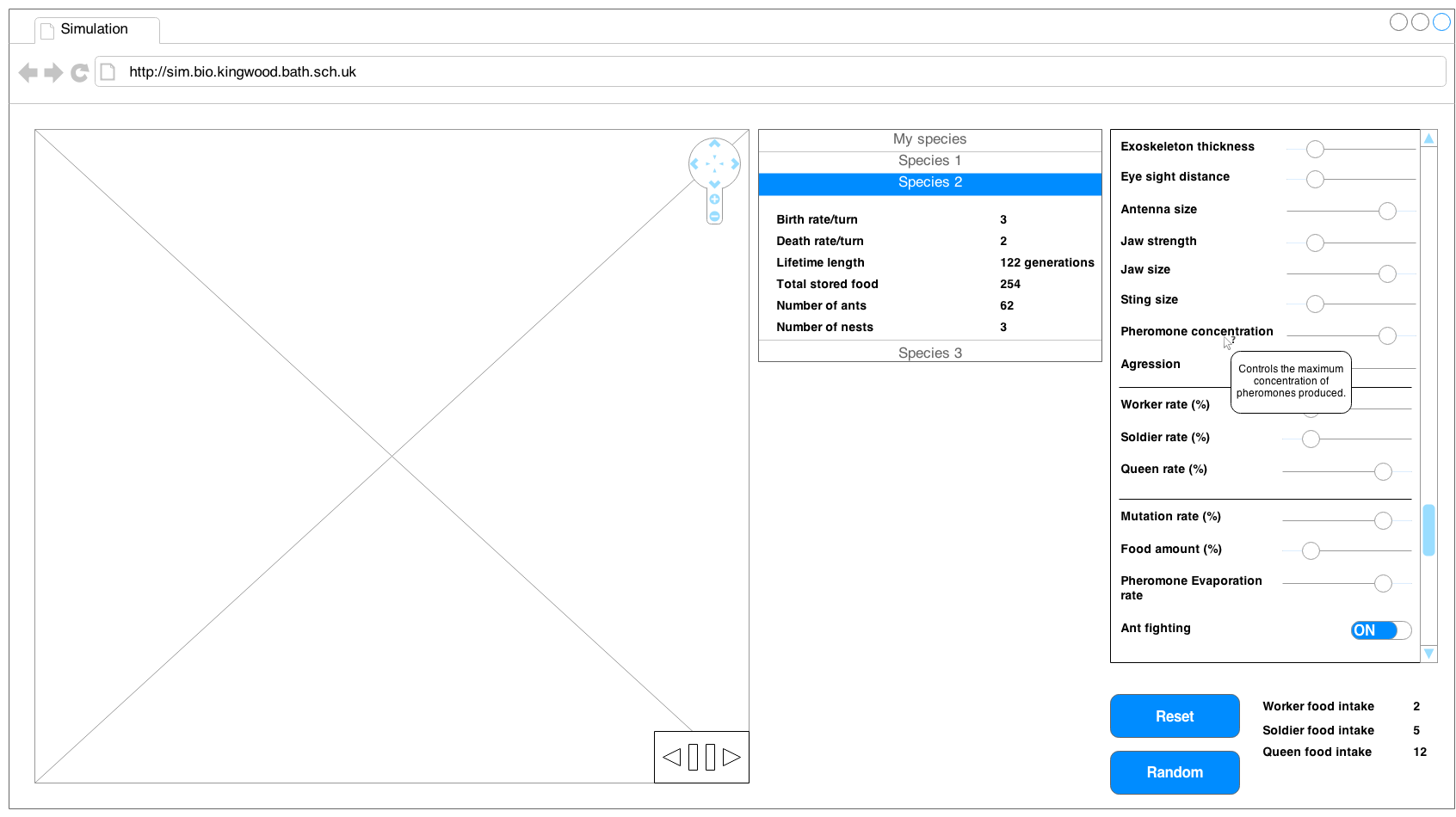
Classes

## User interface design (HCI) rational

### User interaction

This is the expected use of the simulation by the user. The interface will be designed to allow the user to perform the following process as easily as possible:

Interface Mock-up



**6**

**5**

**4**

**3**

**2**

**1**

The interface is split into three main sections. Each section has a single purpose and this will make the layout of the system easy to learn and also will make it more intuitive. The positioning of the panels is such that it mimics the user’s flow through the program. The user will observe the simulation, then look at the data produced by the simulation and finally edit the characteristics in the simulation i.e. flowing left to right across each panel.

#### The simulation panel

The simulation panel is located on the far left of the screen. It will be the first thing the user sees when they start the application. The simulation will start running so that it is obvious that this is the simulation and also provide a point of interest for the user.

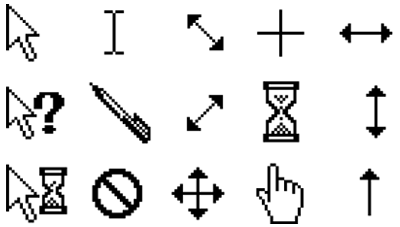
The main item in the simulation panel is the simulation itself. The design of the simulation is stated in the UI sample of planned valid output designs. However there are also two other main items in the panel. These are both controls for the simulation:

* **Time controls** – located in the bottom right of the simulation. These will be used to control how time is displayed in the simulation. It will have three buttons: a pause/play button and on either side a step forward and step backward a single tick. The icons used will be intuitive to the user regarding their function because of their universal use. The colours used will be neutral in comparison with the bright colours of the simulation. This is because the controls should not attract attention away from the simulation. The size of these controls is small compared with the size of the simulation. The reason for this is because the controls are on top of the simulation and big controls would reduce the amount of viewable simulation.
* **Navigation controls** – located in the top right of the simulation. These controls will allow the user to both zoom in and out as well as pan around the simulation. Again the icons chosen are popular in many other applications and synonymous with map navigation. Again these controls will have neutral colours. The position of these controls will be in line with the time controls. The reason for this is so that the user will not have to move their mouse too far to use the controls and also so the user will not have to move their mouse across the simulation.

The user will not only be able to navigate in time using the buttons provided, they will also be able to use a number of shortcuts which will allow them to perform these actions much quicker than hitting a button. The shortcuts will be:

* **Spacebar** – pause/play the simulation. The spacebar is used as it is the biggest button on the keyboard and it very accessible to the user. Also it is used in a number of other applications as the pause button and so the user will already be familiar with the concept of it being the pause button.
* **“f” key** – the “f” key would step forward a single tick in the simulation. “f” is chosen as it is the first letter of “forward” and therefore will be memorable.
* **“b” key** – the “b” key would step backwards a single tick in the simulation. “b” will be used as it is the first letter of “backwards” and so it will be easily memorable for the user. What’s more, the “b” and “f” keys are in close proximity on the keyboard and so it will be easy for the user to press either of these keys.

Navigation will also be able to be done without the use of pressing buttons on the screen. The advantage of this is that it will be much easier and faster for the user to perform these actions. The user will use the arrow keys to navigate. The reason this was chosen is that many games and mapping programs use the arrow keys to pan around a map. The plus and minis (“+” and “-”) keys will be used as a shortcut for zooming in and out. Again the use of these keys is common practice for zooming in many other applications and so the user will intuitively be able to use this function.

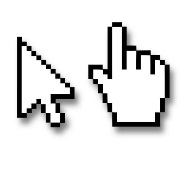
Finally, the mouse will be another way to navigate around the map. Holding the mouse down on a spot and dragging in a direction will be used to pan around the map. And scrolling forward and backward will be used for zooming. To make it obvious that the user can use the mouse to interact with the simulation the cursor of the mouse will change when it is move over the simulation from the default pointer to a navigation cursor.

#### The data panel

The data panel is located in the centre of the screen. The data panel will be used to display information about the species in the simulation. It will be automatically updated as the simulation runs. To make the information more compact, the data about each species will be expectable and contractible. The colour of each species in the table will be the same colour as it appears in the simulation. This will be done so that the user can easily spot which ants correspond to which species and thereby speeding up the fluency with which they use the simulation.

The information displayed about each species will be:

* **Birth rate/turn** – this is the number of ants born each turn for the species.
* **Death rate/turn** – this is the number of ants which die each turn.
* **Lifetime length** – this is the time in number of ticks that the species has been around for.
* **Total stored food** – this is the total amount of food in all nests in the species.
* **Number of ants** – this is the number of ants in the species.
* **Number of nests** – this is the number of nests in the species.

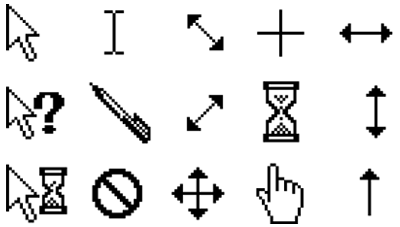
A single species will be capable of being selected by clicking on it in the table. To symbolise a selected species it will be coloured a blue colour which is synonymous with highlighted items in most web browsers. This will help the user to understand what happens when they click on a species from the table. What’s more, the user’s cursor will change to a pointer rather than a default cursor when the user hovers over a species. This will make it obvious to the user that the item is intractable and will change when clicked. The reason the user will know this is because the pointer cursor is used when you hover over a hypertext link on a webpage.

When a species is selected, the simulation will centre on a nest of that species in the simulation and the characteristics of that species will appear in the configuration panel. This allows the user to easily see more information about the species i.e. the location of its nests in the simulation and its characteristics.

#### The configuration panel

The configuration panel is located on the far right of the screen. It contains the characteristics of the selected species and allows them to be edited by using sliders. Sliders are used as they force type and range validation on the input. This is extremely useful as it decreases the amount of errors and warnings which the user will have to see (errors and warnings are bad as they disrupt the user experience).

The characteristics will be arranged in a logical order i.e. will group related characteristics together. This will allow the user to more easily find characteristics quickly and explore how similar characteristics affect the simulation.

As the impacts of the characteristics are not immediately obvious to the user a hover text will be provided i.e. when the user hovers over a characteristic a description of the characteristic will appear. In order that the user will use this feature the cursor will change to a help cursor when the user hovers over a characteristic.

Aside from the characteristics, controls for choosing characteristics will also be available. These controls are:

* **Random** – this will generate completely random characteristics. This is useful to the user as it easily allows them to see a wide range of the types of species which may appear in the simulation.
* **Reset** – this will reset all the characteristics of a species to the default values. This is clearly useful as it allows the user to mess around with characteristics to see their effect and then easily reset them all without refreshing the page.

Finally, data about the consequences of the characteristics values picked is displayed at the bottom of the panel. The information displayed will be:

* **Worker food intake** – displays the amount of food it will take to generate a worker ant.
* **Soldier food intake** – displays the amount of food it will take to generate a worker ant.
* **Queen food intake** - displays the amount of food it will take to generate a worker ant.

This is useful to the user as it allows them to instantly see feedback from the change in characteristics i.e. as a change may take some time to be noticeable in the simulation.

#### Colour scheme and fonts

There will only be a single font used in the entire interface. This is to increase consistency in the page. The font used will be Helvetica. Helvetica will be used as it is available on most modern operating systems and also provides a clean and comfortable look for the page compared with more formal serif fonts like Times New Roman.

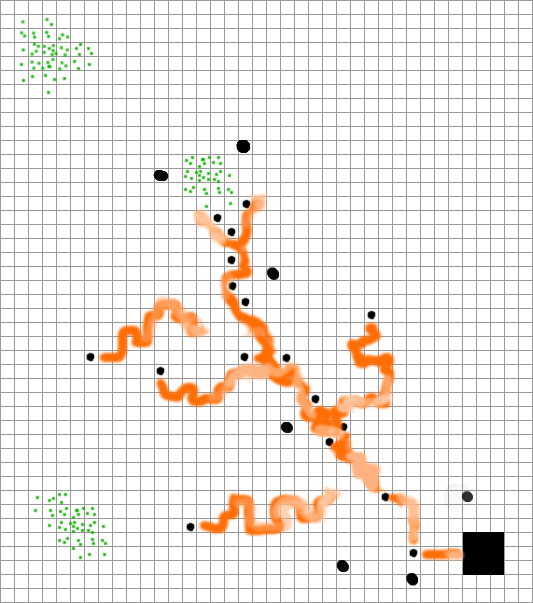
The colours used in the interface will be neutral black and white in each of the panels. The reason for this is to draw attention to the colours in the simulation which should be the main point of focus of the page. However a bright background around the panels may be used to make the page look fun.

## UI sample of planned data capture and entry designs

Data will be collected when the user edits a characteristic. Each characteristic is listed in the characteristics panel. Ranges are used to enter the data into the simulation, the reasons for this is to reduce the number of validation errors and warnings the user will have to see. This is because range inputs limit the input to a certain type and also between a certain ranges of values.

*Note*: For UI for data capture see Configuration panel in Interface Mock-ups

## N:\IMG_20140310_130914.jpgUI sample of planned valid output designs



#### Colours

Each species in the simulation will have a unique colour. This colour will be randomly generated. This will mean that the simulation will be interesting as it will have many different colours. All ants, nests and pheromones of the species will have this same colour.

Pheromones in the simulation will be displayed in the colour of the ant species. Pheromones with a higher concentration or newly laid pheromones will be more vivid showing that they are more active and recent, while older more stagnant trails will be very light, as to not attract attention from the user.

Food is displayed as green - a colour often associated with leaves, grass etc. (which are the food ants eat in real life). This will be intuitive to the users as many games use the colour green to symbolise food.

#### Shapes

In the simulation a number of different shapes will be used to differentiate between different items. This is needed as ants, nests and pheromones from the same species will be the same colour and so to determine which is which their shape is used.

Worker ants will be a square. A square, as opposed to a more lifelike representation of an ant, is used as it will decrease clutter on the screen and allow the user to focus more on the behaviour of the ant rather than how it looks. When an ant is carrying food it will have a smaller green square on its back. This will be useful as it will show to the user which ants are carrying food and so they can more easily monitor these ants’ behaviour. The square is green as it shows that they are carrying food, which is also green.

Soldier ants will be displayed as squares also. However soldier ants will have a white cross on them. The cross will turn red when they are attacking another ant. Red was chosen as it is often seen as a colour associated with anger and therefore attacking/fighting will be logically assumed. The change in colour of the cross is useful as it tells the user extra information about the current goal of the ant or why they are behaving in a particular way e.g. why are they following another ant.

The queen ant differs from both the soldier ant and the worker ant in its shape. The queen ant will be a circle. This is because queen ants are of particular importance as they have the ability to create a new nest and therefore will be of interest to the user. Furthermore queen ants will be much less likely to be born and so it will be a rare event when one is created. To maximise the chance that the user will be able to spot the queen ant its shape will be different from all of the other ants.

A nest in the simulation will be a large square. This is chosen as due to its size it will be immediately obvious to the user where it is.

Food in the simulation will be displayed as a collection of dots. The number of dots will represent the amount of food in the cell. This is done so that it will be easy for the user to estimate the concatenations of food in particular regions of the map just by glancing at the area.

Pheromones will be displayed as long trails behind ants. This is much like in real life, however trails are invisible to the naked eye. The trails will be visible so that the user can easily see the areas of high activity by looking for the areas with the most concentrated pheromones. It will also be nice for the user to be able to see the trails of where ants have been moving, and also see what trails are influencing the movement of ants.

## Description of measures planned for security and integrity of data

There will be no personal or sensitive information recorded or entered into the system and because of this there is no need to have a password protected system.

To protect the integrity of the application itself it will be hosted on a school website and mirrored on a number of other computers. This means that it will be difficult to lose the application even if a server is destroyed as it will be backed up elsewhere and easily recoverable.

## Overall test strategy

As part of the development strategy, methods from test driven development will be used to create parts of the application. This means that for some parts of the project unit tests will be written which will be used to aid development. A testing framework will be used during the development of the project to generate and automatically tests these tests.

White box test Ant, worker, queen, soldier and nest classes because methods of these classes will primarily contain logic and so it is important to test each non trivial logical path to ensure that the procedures perform correctly. Utilities functions i.e. functions which do not belong to any class but provide useful actions which can be applicable to many situations, will be black box unit tested. This is because these functions will be more mathematically oriented rather than logically e.g. the angle between two points.

As the tool will be used outside the classroom i.e. it is set for homework, it is important for the toll to work on multiple platforms. So it will be tested on multiple platforms (operating systems and browsers) to make sure that it can be used outside the classroom. Due to the large numbers of possible operating systems and web browser combinations it is necessary to limit these tests to only the most popular choices.

The simulation will be very intensive for large numbers of ants, so it will be necessary to carry out a stress test to check that the simulation will be able to cope on the hardware available in the class room.

User interface functional testing will also be carried out to test that the user interface works as expected. This includes testing that buttons perform the correct actions and that inputs and outputs are displayed correctly.

Finally, it will be necessary to test the simulation against real life biological models. As the simulation will model ants it should be able to be compared to real life ants and produce similar results/patterns. This is needed to show that the simulation truly models ants.

# Section 3: Technical Solution

All evidence of the technical solution has been added in appendix C.

# Section 4: System Testing

## Test Plan

The strategy chosen to test the solution is to test the basic functionality of functions which are used throughout the application i.e. functions in utilities.js. These will be tested with automatic unit tests which use black box testing to check the output is correct for different inputs. Drawing aspects of the application will be tested by checking that the output is as expected. This will be done for all classes with a \*.draw function. The methods in the ant, worker, soldier and queen classes will be tested with black box unit tests, white box unit tests and integration testing. As the majority of these methods are logic based i.e. they have lots of different conditions rather than calculating a specific thing, it is easiest to test each non obvious path through the method.

Certain modules will not need testing due to their simplicity. The canvas module does not require testing as it is basically a reimplementation of calls to the HTML canvas API so that they can be used in a shorthand version. What’s more the map module only contains very basic functions and so does not need to be tested.

As the simulation may be potentially very processer intensive, it is important that it will work on the machines provided by the school. Therefore a stress test of the simulation will take place to make certain that the application will be able to run at an acceptable level under normal operating conditions.

Due to the nature of the simulation, a comparison between real life biological process and those simulated will be compared. This is to check that the simulation will act as a good comparison to the real life biological system.

It is difficult for the user to enter erroneous data into the simulation due to the use of sliders for inputs. And the simulation will not generate erroneous data internally and therefore there is a lack to erroneous testing because of this.

To test that the system performs as expected it will be run at a much slower pace compared with how it will be normally operating. This is done so that system can be acutely observed.

*Note*: Some functions are common among multiple classes such as die, updateGoal, doTask, draw and update. These functions do not require testing as they are intuitive and don’t contain any complex logic or operations.

*Note*: Dummy objects for used for some tests so that unit tests do not rely on other classes or data to operate. This helps to make tests independent i.e. they can be run with only a single module or class imported.

## Test data

### Utilities.js - unit tests

| **Test** | **Function** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | validateDirection | Returns an angle in the range 0 ≤ θ < 2π | **Typical** | 5π | 1π | PASS |
| 2 | **Typical** | 0 | 0 | PASS |
| 3 | **Typical -** Negative value | -2π | 0 | PASS |
| 4 | angleTo | Returns the correct angle to get from coord to target | **Typical** | coord : {x : 5, y : 5}, target : {x : 7, y : 5} | ½ π | PASS |
| 5 | **Typical** | coord : {x : 5, y : 5}, target : {x : 7, y : 7} | 3π/4 | PASS |
| 6 | **Boundary –** Coordinate on top of ant i.e. navigating to its own position | coord : {x : 5, y : 5}, target : {x : 5, y : 5} | π/2 | PASS |
| 7 | **Typical -** wrapped direction i.e. going off map to get to position | {x : 8, y : 8}, target : {x : 1, y : 1} | 3π/4 | FAIL- revised and passed with function update |
| 8 | **Typical -** wrapped direction | coord : {x : 7, y : 8}, target : {x : 3, y : 2} | ≈ 3.38657 | PASS |
| 9 | boundary | Returns the correct warped coordinate | **Typical** | {x : 4, y : 2} | {x : 4, y : 2} | PASS |
| 10 | **Boundary -** Both x and y boundaries | {x : 10, y : 10} | {x : 0, y : 0} | PASS |
| 11 | **Typical -** X-axis boundary only | {x : 10, y : 4} | {x : 0, y : 4} | PASS |
| 12 | **Typical -** Y-axis boundary only | {x : 7, y : 10} | {x : 7, y : 0} | PASS |
| 13 | clone | Produces an exact clone of an object | **Typical** | {a : [1, 2, 3], b : {c : 4, d : '5'}, e : {f : {g : 6}}, h : [['end', 'of', ['object']]]} | {a : [1, 2, 3], b : {c : 4, d : '5'}, e : {f : {g : 6}}, h : [['end', 'of', ['object']]]} | PASS |
| 14 | Produces a copy of the object i.e. not by reference | **Typical** | let b = {a : 1} | {a : 1} != b | PASS |
| 15 | coordToIndex | Returns the correct index for a specific coordinate | **Typical** | {x : 0, y : 0} | 0 | PASS |
| 16 | **Typical** | {x : 5, y : 2} | 25 | PASS |
| 17 | distance | Returns the correct distance | **Boundary -** Both the wrapped distance and the regular distance are the same | coord1 : {x : 0, y : 0}, coord2 : {x : 0, y : 0} | 0 | PASS |
| 18 | **Typical** | coord1 : {x : -2, y : 1}, coord2 : {x : 1, y : 5} | 5 | PASS |
| 19 | getBlock | Returns a block of cells the correct size | **Boundary -** 0 size | coord : {x : 5, y : 5}, size : {width : 0, height : 0} | A single block centred at {x : 5, y : 5} | PASS |
| 20 | **Typical** | coord : {x : 5, y : 5}, size : {width : 3, height : 3} | A 7 wide and 7 tall block centred at {x : 5, y : 5} | PASS |
| 21 | **Typical -** Non square | coord : {x : 5, y : 5}, size : {width : 1, height : 3} | A 3 wide and 7 tall block centred at {x : 5, y : 5} | PASS |
| 22 | Returns a block of cells the correct position | **Typical** | coord : {x : 1, y : 3}, size : {width : 2, height : 2} | A 5 wide and 5 tall block centred at {x : 1, y : 3} which wraps to the other side of the map | PASS |
| 23 | **Boundary -** Wrapped in all quadrants | coord : {x : 0, y : 0}, size : {width : 1, height : 1} | A 3 wide and 3 tall block centred at {x : 0, y : 0} which wraps to every corner of the map | PASS |
| 24 | getCellCoord | Returns the correct cell for a particular coordinate | **Typical** | {x : 7.2, y : 3.6} | {x : 7, y : 3} | FAILED - revised and passed with function update |
| 25 | getSector | Returns sector of correct radius | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : 0, angle : 2π | A circle of radius 15 centred at {x : 25, y : 25} | PASS |
| 26 | **Typical** | coord : {x : 25, y : 25}, radius : 6, direction : 0, angle : 2π | A circle of radius 6 centred at {x : 25, y : 25} | PASS |
| 27 | Returns sector at the correct angle | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : 0, angle : π | A sector of radius 15 centred at {x : 25, y : 25} with an angle of π i.e. a semi-circle | PASS |
| 28 | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : 0, angle : π/4 | A sector of radius 15 centred at {x : 25, y : 25} with an angle of π /4 | PASS |
| 29 | Returns sector in correct direction | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : π, angle : π /4 | A sector of radius 15 centred at {x : 25, y : 25} with an angle of π/4 pointing downwards | PASS |
| 30 | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : 6π/ 4, angle : π /4 | A sector of radius 15 centred at {x : 25, y : 25} with an angle of π/4 pointing along the centre line of the 4th and 3rd quadrants | PASS |
| 31 | **Typical** | coord : {x : 25, y : 25}, radius : 15, direction : 3.657, angle : π/4 | A sector of radius 15 centred at {x : 25, y : 25} with an angle of π/4 pointing along the centre line of 3.657 radians | PASS |
| 32 | indexToCoord | Returns the correct coordinate from a specific index. | **Typical** | 25 | {x : 5, y : 2} | PASS |
| 33 | **Typical** | 0 | {x : 0, y : 0} | PASS |
| 34 | randColour | Generates random colours. *Note*: Each is run multiple times to test that always produces valid hexadecimal colour in range #000000 to #FFFFFF. | **Typical** | N/A | A hexadecimal colour | PASS |
| 35 | randFloat | Returns a value within a specific range. *Note*: Each is run multiple times to test that always produces value in range. | **Typical** | {min : -1, max : 1} | In range -1 to 1 inclusive | PASS |
| 36 | **Typical** | {min : -0.1, max : 0.1} | in range -0.1 to 0.1 inclusive | PASS |
| 37 | **Typical** | {min : 0, max : 2} | In range 0 to 2 inclusive | PASS |
| 38 | randInt | Returns a value within a specific range. *Note*: Each is run multiple times to test that always produces value in range. | **Typical** | {min : 0, max : 100} | In range 0 to 100 inclusive | PASS |
| 39 | **Boundary -** No range of values | {min : 0, max : 0} | 0 | PASS |
| 40 | **Typical** | {min : -20, max : 20} | In range -20 to 20 inclusive | PASS |
| 41 | randProperty | Returns a random property from an object literal. *Note*: ran multiple times to test that not the same property each time. | **Typical** | {a : 0, b : 1, c : 2} | either 'a', 'b' or 'c' (random each time) | PASS |
| 42 | scaleCoord | Scales coordinates correctly. | **Typical** | {x : 4, y : 2} | {x : 23, y : 12} | PASS |

#### utilities.js unit tests – Evidence

| **Test** | **Evidence** | **Comment** |
| --- | --- | --- |
| 1 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\validateDirection - returns the correct value.PNG |  |
| 2 |  |
| 3 |  |
| 4 |  | Cells highlighted in blue are the location the ant is trying to get to. The cell in red is the location of the ant. The arrow represents the direction the ant will move in. |
| 5 |  |
| 6 |  |
| 7 |  | The image on the left is the passed test. The image on the right is the failed test. The image on the right failed as it did not point in the direction which would lead to the shortest distance between the two items. This was because it didn’t take the wrapped position into account |
| 8 |  |  |
| 9 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\boundary - returns correct wraped coordinate 1.PNG | Cells highlighted in blue are the selected cells |
| 10 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\boundary - returns correct wraped coordinate 2.PNG |
| 11 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\boundary - returns correct wraped coordinate 3.PNG |
| 12 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\boundary - returns correct wraped coordinate 4.PNG |
| 13 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\clone - by reference.PNG |  |
| 14 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\clone - exact clone.PNG |  |
| 15 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\indexToCoord - returns correct values.PNG |  |
| 16 |  |
| 17 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\distance - returns correct values.PNG |  |
| 18 |  |
| 19 |  | Cells highlighted in blue are the selected cells |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\getCellCoord - returns correct coordinate for a point 1.PNG\\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\getCellCoord - returns correct coordinate for a point - FAIL.PNG | The image on the left is the passed test while the image on the right is the failed test. Notice that the red dot represents where the exact coordinate of the ant is. While the blue square highlights the cell which the coordinate mostly lies in |
| 25 |  | Highlighted cells in blue which are visible. The ant’s location is highlighted in red |
| 26 |  |
| 27 |  |
| 28 |  |
| 29 |  |
| 30 |  |
| 31 |  |
| 32 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\indexToCoord - returns correct values.PNG |  |
| 33 |  |
| 34 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\randColour - Tests if generates random colours (3 times).PNG | A sample of colours shown. All colours lie in range and are different |
| 35 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\randFloat - returns a value within a specific range.PNG | Ran multiple times to make sure rand number generated is always in the range |
| 36 |
| 37 |
| 38 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\randInt - returns a value within a specific range.PNG | Ran multiple times to make sure rand number generated is always in the range |
| 39 |
| 40 |
| 41 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\randProperty - returns a random property from an object literal.PNG | Ran multiple times to make sure picks a random property from the object. This is a sample of 3 runs |
| 42 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\utilities\scaleCoord - scales correctly.PNG |  |

#### utilities.js unit tests – Failed tests

#### Test

Test 7failed as angleTo did not pick the angle which would lead to the shortest path to the target coordinate. The correct angle would have pointed of the map, as the wrapped distance was shorter than the normal distance. This was because angleTo did not take wrapped directions into account in its original form:

**function** angleTo(coord, target) {

**var** dx = target.x - coord.x;

**var** dy = target.y - coord.y;

**return** Math.atan2(dy, dx) + Math.PI/2;

}

The new function takes into account wrapping around the map to get the angle of the shortest path. It uses a similar algorithm to the one in the distancefunction.

**function** angleTo(coord, target) {

**if** (GRID\_SIZE.width **-** Math.abs(target.x **-** coord.x) **>** Math.abs(target.x **-** coord.x)) {

        dx **=** target.x **-** coord.x;

    } **else** {

        dx **=** GRID\_SIZE.width **-** (target.x **-** coord.x);

    }

**if** (GRID\_SIZE.height **-** Math.abs(target.y **-** coord.y) **>** Math.abs(target.y **-** coord.y)) {

        dy **=** target.y **-** coord.y;

    } **else** {

        dy **=** GRID\_SIZE.height **-** (target.y **-** coord.y);

    }

**return** Math.atan2(dy, dx) **+** Math.PI **/** 2;

}

#### Test 24

Test 24 originally failed due to the getCellCoord function using Math.round rather than Math.floor in its operations, this meant that if the fractional part of x or y was 0.5 or greater the function would be out by one cell.

### Ant.js – unit tests

| **Test** | **Function** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Ant.scan | Adds all ants within viewing distance to ant.itemsInView.ants | **Typical** | No ants in view | this.itemsInView.ants.length = 0 i.e. cannot see any ants in view | PASS |
| 2 | **Typical** | Multiple ants within viewing distance | this.itemsInView.ants.length = 4 | PASS |
| 3 | Adds food in viewing distance to ant.itemsInView.food | **Typical** | No food in view | this.itemsInView.food.length = 0 i.e. cannot see any food in view | PASS |
| 4 | **Typical** | Multiple pieces of food in view | this.itemsInView.food.length = 3 | PASS |
| 5 | Ant.secrete | Adds pheromone of correct concentration | **Typical** | No pheromones to start | A new pheromones of concentration 0.5 | PASS |
| 6 | **Typical -** Adding pheromone | A pheromone of the same species | A new pheromones of concentration 0.9 i.e. 0.4 + 0.5 | PASS |
| 7 | **Typical -** Not adding pheromones as different species | A pheromone of a different species | A new pheromones of concentration 0.5 | PASS |
| 8 | **Typical -** Doesn’t exceed MAX\_PHEROMONE\_CONCENRATION | A pheromone of the same species | A new pheromones of concentration 1 | PASS |
| 9 | Ant.atNest | Ant is on nest or not | **Typical** | not on the nest | FALSE | PASS |
| 10 | **Typical** | On the nest | TRUE | PASS |
| 11 | Ant.seeNest | Can see nest when in range | **Typical** | Cannot see the nest | FALSE | PASS |
| 12 | **Typical** | Nest within view | TRUE | PASS |
| 13 | Ant.smell | Adds all pheromones within range to pheromonesInRange | **Typical** | No pheromones in rang | this.pheromonesInRange.length = 0 i.e. cannot smell any pheromones in view | PASS |
| 14 | **Typical -** Shouldn't read pheromone it’s on | 4 Pheromones in range with one on top of the ant | this.pheromonesInRange.length = 3 | PASS |
| 15 | Ant.takeFood | Takes correct amount of food | **Typical** | void(0) i.e. No food | 0 | PASS |
| 16 | **Typical** | A single piece of food of amount 3 | After three runs of Ant.takeFood food = void(0) | FAIL - PASSED after re writing function |
| 17 | Ant.wonder | Ant picks correct direction given some amount of pheromones | **Typical** | No pheromones in view | this.direction === this.prioritizeDirection | PASS |
| 18 | **Typical** | A single pheromone | Ant is pointing at the pheromone | PASS |
| 19 | **Typical** | Multiple pheromones | Ant is pointing slightly upwards to the right | PASS |

#### Ant.js unit tests – Evidence

| **Test** | **Evidence** | **Comment** |
| --- | --- | --- |
| 1 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.scan - tests if an ant can see other ants.PNG |  |
| 2 |  |
| 3 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.scan - tests if an ant can see food.PNG |  |
| 4 |  |
| 5 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.secrete - tests ant drops correct pheromones.PNG |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.atNest - tests if an ant is at the nest.PNG |  |
| 10 |  |
| 11 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.seeNest - tests if an ant can see a nest.PNG |  |
| 12 |  |
| 13 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.smell - tests if ant can smell pheromones.PNG |  |
| 14 |  |
| 15 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.takeFood - test ant takes food.PNG  \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.takeFood - test ant takes food - FAIL.PNG | Top image shows both tests passing. The bottom image shows one test has failed. Due to an undefined property. See *Ant.js unit tests – Failed tests* section for more details |
| 16 |
| 17 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.wonder - test ant follows pheromones 1.PNG | A cell highlighted in blue represents a pheromone. The red cell represents the ant. The red arrow represents the direction the ant is facing. Note that all pheromones are of the same concentration i.e. the same shade of blue |
| 18 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.wonder - test ant follows pheromones 2.PNG |
| 19 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Ant\Ant.wonder - test ant follows pheromones 3.PNG |

#### Ant.js unit tests – Failed tests

#### Test 16

Test 16 failed originally due to an error in the Ant.takeFood function.

**if** (**this**.isFood(food))

food.removeFromMap();

There was a missing not in the stamen which meant that the food would be removed after one piece of food was taken. Here is the fixed version:

**if** (**!this**.isFood(food))

food.removeFromMap();

### Nest.js - unit tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Function** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| 1 | Nest.reproduce | Produces the correct probabilities | **Typical** | Soldier probability = 1, worker and queen probabilities = 0. Run 10,000 times to check probabilities. | Only soldier ants produced | PASS |
| 2 | **Typical** | Soldier, Worker and Queen probabilities are all 1. Run 10,000 times to check probabilities. | A 1/3 chance of each type of ant | PASS |
| 3 | **Boundary**– sum will be 0 therefore dividing by 0 should produce null | Soldier, Worker and Queen probabilities are all 0. Run 10,000 times to check probabilities. | No ants created | PASS |
| 4 | Nest.calcSpeciesCost | Calculates the correct sum of each characteristic cost | **Typical** | species.chars = {speed : 0.51, stingSize : 3, eyesight : 5} | 337 | PASS |
| 5 | **Boundary**– No properties in CHARS | species.chars = {} i.e. no properties | 0 | PASS |

#### Nest.js unit tests – Evidence

|  |  |  |
| --- | --- | --- |
| **Test** | **Evidence** | **Comment** |
| 1 | C:\Dropbox\projects\Ant-Simulation\writeup\assests\Maintainance\Testing\Passed Nest.reproduce test - Human anylis.PNG | This test relies on probabilities; it therefore requires human input to determine if the probabilities were close to match the theoretical result |
| 2 |
| 3 |
| 4 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Nest\Nest.calcSpeciesCost - correct sum.PNG |  |
| 5 |  |

### Worker.js - Integration tests

| **Test** | **Action** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Searching for food | Tests an ant will search for food | **Typical** | A single piece of food on the map | The ant to wonder around, scanning its surroundings and interpreting the items in view for food targets. When it spots food, its goal updates to getting food | PASS |
| 2 | Retrieving food | Tests an ant will retrieve food which it has targeted correctly.*Note*: tests canCarry and useFood functions | **Typical** | An ant facing a single piece of food which is in viewing distance | The ant will move towards the food until it is on top of it. The ant will determine how to best use the food i.e. eat or carry. The ant will carry out the action associated with its choice. A single piece of food should be removed from the food source and either added to the ants carrying amount or added to its health | PASS |
| 3 | Dropping food | Tests an ant will drop food to its own nest.*Note*: tests depositeFood and dropFood functions | **Typical** | An ant carrying 3 pieces of food and a nest of the same species | The ant to wonder around until it sees the nest. Then move directly towards the nest. Upon reaching the nest, spend 3 ticks until all food is gone. The ant’s goal will then update to finding food. All the while pheromones are being laid | PASS |
| 4 | draw | Test that the ant will be drawn correctly | **Typical** | Ant not carrying food | Black square to be drawn | PASS |
| 5 | **Typical** | Ant carrying food | Black square with smaller square inside it | PASS |

#### Worker.js integration tests – Evidence

| **Test** | **Evidence** | **Comment** |
| --- | --- | --- |
| 1 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Worker\Worker.findingFood - Worker finding Food 2.PNG\\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Worker\Worker.findingFood - Worker finding Food.PNG | The first image shows the output of the test. It is logging events the ant is going through when searching for food.   * Searching for food means the ant is using the wondering function to walk around * Found food target means that the ant has spotted a food source * The updating the goal shows that the ant successfully updates its goal to the next task   The second image shows a visual sample of the output. It is a much smaller environment. Food is green, black is the ant and blue is what the ant can see |
| 2 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Worker\Worker.draw - Drawn with food.PNG | This shows that the ant is carrying food as it is being drawn with a square representing it holding food |
| 3 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Worker\Worker.draw - Drawn with out food.png | A sample from the test showing the ant has dropped the food, as it has no longer being drawn with a square on it. *Note*: The nest is not drawn in the picture however the test creates a 1x1 mock nest on the square the ant is standing |
| 4 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\Worker\Worker.draw - Drawn with out food.png |  |
| 5 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\Worker\Worker.draw - Drawn with food.PNG |  |

| **Test** | **Action** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Guarding nest | Tests an ant will guard its nests correctly | Typical | A single soldier ant and its nest | The ant will search for its nest, when it finds its nest it will move certain distance away, stop and start turning around | PASS |
| 2 | Guarding food | Tests an ant will guard food correctly | Typical | A single soldier ant and a single source of food | The ant will search for the food. When the ant sees the food, it will stop and start turning around | PASS |
| 3 | Attacking other ants | Tests an ant will attack ants it has targeted correctly. *Note*: tests pickTarget, follow and attack functions | Typical | A single soldier ant and a single worker ant of different species | [For solider ant only]The ant will wonder around, when it sees the other ant it will target the ant. When the ant is targeted it will begin to follow the ant until it is range to attack. The ant will then attack the other ant reducing its health until death. This will continue to happen until the other ant is dead | PASS |
| 4 | Soldier.soldiersInView | Test that soldier in view returns true when there are soldiers in view | Typical | No ants in view | Returns false i.e. no soldiers in view | PASS |
| 5 | Typical | Multiple ants within viewing distance which are both workers and soldiers. | Returns true. | PASS |
| 6 | Typical | A single ant within viewing distance which is not a soldier | Returns false | PASS |
| 7 | Typical | A single ant within viewing distance which is a soldier. | Returns true | PASS |
| 8 | Test that soldier in view correctly determines if there are friendly (i.e. of same species) ants in view | Typical | A solider ant of the same species | Returns true | PASS |
| 9 | Typical | A soldier ant of a different species | Returns false | PASS |

### Soldier.js – integration tests

#### Soldier.js integration tests - Evidence

| **Test** | **Evidence** | **Comment** |
| --- | --- | --- |
| 1 | N/A | Program was stepped through in debugger and variables were watched to make sure they were producing the correct values. *Note*: test code can be found in appendix for more information on how test works |
| 2 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Soldier\Soldier.guardFood - Ant guards food.PNG | Green represents a single piece of food and the black square with a cross on it is an ant |
| 3 | N/A | Program was stepped through in debugger and variables were watched to make sure they were producing the correct values. *Note*: test code can be found in appendix for more information on how test works |
| 4 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Soldier\Soldier.soldiersInView - test works when ants in view.PNG |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Soldier\Soldier.soldiersInView - test works when ants in view of different species.PNG |  |
| 9 |  |

### Species.js – unit tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Function** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| 1 | Species.mutate | Tests that characteristics can be mutated | **Typical** | Default characteristic values | A random characteristics value is altered or the same characteristics are returned | PASS |

#### Species.js unit tests – Evidence

|  |  |  |
| --- | --- | --- |
| **Test** | **Evidence** | **Comment** |
| 1 | N/A | Ran mutate multiple times (as there is a chance it will not mutate due to probabilistic nature of function) |

### Food.js - unit tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Action** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| 1 | Food.draw | Tests the piece of food is correctly drawn for different amounts of food | **Typical** | Amount 1 | Alpha value of 0.2 | PASS |
| 2 | **Typical** | Amount 5 | Alpha value of 1 | PASS |
| 3 | **Typical** | Amount 0 | Alpha value of 0 i.e. not visible | PASS |
| 4 | Food.grow | Tests that food will grow the correct amount | **Typcial** | No food around growing food piece | 8 new food pieces around growing block | PASS |
| 5 |  | Food around growing food piece | All food pieces amount increases by 1 | PASS |

#### Food.js unit tests – Evidence

| **Test** | **Evidence** | **Comment** |
| --- | --- | --- |
| 1 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Food\Food.draw - drawing piece of food correctly 1.png |  |
| 2 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Food\Food.draw - drawing piece of food correctly 2.png |  |
| 3 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Food\Food.draw - drawing piece of food correctly 3.png | From console output it is known that the piece of food was drawn however it has an alpha value of zero and therefore nothing to see |
| 3 | N/A | Console output showed 6 new pieces of food where created |
| 4 | N/A | From debug console food amount of pieces where monitored and increased by one |

### FoodSystem.js – unit tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Function** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| 1 | FoodSystem.addFood | Tests that food is randomly places. *Note*: Ran multiple times to check placement is random each time | **Typical** | N/A | Random placement of food throughout the map | PASS |
| 2 | FoodSystem.addFoodBlob | Tests that the blob created is of the correct size | **Typical** | Food blob radius = 5 | A green circle radius 5 which is darkest in the middle and gets brighter the further out it goes | PASS |
| 3 | **Typical** | Food blob radius = 25 | A green circle radius 25 which is darkest in the middle and gets brighter the further out it goes | PASS |
| 4 | **Typical** | Food blob radius = 0 | A green circle radius 0 which is darkest in the middle and gets brighter the further out it goes | PASS |

#### FoodSystem.js unit tests – Evidence

| **Test** | **Evidence** | **Comment** |
| --- | --- | --- |
| 1 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\FoodSystem\FoodSystem.addFood - test adds random looking food.PNG\\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\FoodSystem\FoodSystem.addFood - test adds random looking food 3.PNG\\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\FoodSystem\FoodSystem.addFood - test adds random looking food 2.PNG | The evidence is three images showing three different random looking placements of food blobs. The test was run multiple times; these three are only a sample of the results |
| 2 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\FoodSystem\FoodSystem.addFoodBlob - correct radius 1.PNG |  |
| 3 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\FoodSystem\FoodSystem.addFoodBlob - correct radius 2.PNG |  |
| 4 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\FoodSystem\FoodSystem.addFoodBlob - correct radius 3.PNG | The console shows that a food blob was created however was of zero size and therefore was not drawn |

### NestPiece.js – integration tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Action** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| 1 | Draw | Tests that a nest piece is correctly drawn onto the map | **Typical** | A single nest piece | A black square | PASS |

#### NestPiece.js integration tests – Evidence

|  |  |  |
| --- | --- | --- |
| **Test** | **Evidence** | **Comment** |
| 1 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\NestPiece\NestPiece.draw - draws nest piece correctly.png |  |

### Pheromone.js – integration tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Action** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| 1 | draw | Tests that a pheromone is correctly drawn for different concentrations | **Typical** | Amount 0.5 | Alpha value of 0.5 | PASS |
| 2 | **Typical** | Amount 0 | Alpha value of 0 i.e. not visible | PASS |
| 3 | **Typical** | Amount 1 | Alpha value of 1 | PASS |

#### Pheromone.js integration tests – Evidence

| **Test** | **Evidence** | **Comment** |
| --- | --- | --- |
| 1 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Pheromone\Pheromone.draw - test pheromone are draw correctly 1.png |  |
| 2 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Pheromone\Pheromone.draw - test pheromone are draw correctly 2.png | Console shows that a pheromone was drawn however its alpha value is zero. In a the real simulation, the pheromone will be instantly removed in the next tick |
| 3 | \\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\app\tests\Pheromone\Pheromone.draw - test pheromone are draw correctly 3.png |  |

### Queen.js – integration tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Action** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| 1 | Draw | Tests that a queen ant is correctly drawn onto the map | **Typical** | A single queen ant | A black circle | PASS |

#### Queen.js integration tests – Evidence

|  |  |  |
| --- | --- | --- |
| **Test** | **Evidence** | **Comment** |
| 1 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\Queen\Queen.draw - ant is drawn correctly.png |  |

### Controls.js – unit tests

*Note*: Many functions from this module are tested in *user interface functionality testing* section.

| **Test** | **Function** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | updateValue | Tests that correctly updates CHARS from the html value field | **Typical** | Speed = 0.5 | Update button changes colour. Value HTML field updates. Characteristics value in CHARS updates. Ant species costs update | PASS |
| 2 | **Erronous** | QueenStepsMin = 10  QueenStepsMax = 5 | Raise Error #1 i.e. queenStespMin is greater than queenStepsMax | PASS |
| 3 | **Erronous** | workerFoodCost = 1000 | Raise Warning #1 i.e. worker food cost is greater than species cost | PASS |
| 4 | **Erronous** | soldierFoodCost = 1000 | Raise Warning #2 i.e. soldier food cost is greater than species cost | PASS |
| 5 | **Erronous** | queenFoodCost = 1000 | Raise Warning #3 i.e. queen food cost is greater than species cost | PASS |
| 6 | **Boundary** | QueenStepsMin = 10  QueenStepsMax = 10 | No Error message | PASS |
| 7 | **Boundary** | workerFoodCost = 2000 | No warning message | PASS |
| 8 | **Boundary** | soldierFoodCost = 2000 | No warning message | PASS |
| 9 | **Boundary** | queenFoodCost = 2000 | No warning message | PASS |
| 10 | createInputType | Test that creates a valid HTML input | **Typical** | Characteristic : CHARS[speed]  Prop : “speed” | A new input element | PASS |
| 11 | **Typical** | Non editable characteristic  Characteristic: CHARS[speed]  Prop : “speed” | A new input element with disabled attribute | PASS |
| 12 | createInput | Test that creates a valid container for the input | **Typcial** | Characteristic: CHARS[speed]  Prop : “speed” | Creates a new container for the html input. This should include a label, the input its self and a value element | PASS |
| 13 | createSpeciesData | Test creates a valid HTML elements | **Typcial** | A mock species | Creates a title for the species. Creates a colour row with label and a colour. Creates a number of ants row with label and value. A number of nests row with label and value. An Amount of food row with a label and a value | PASS |

#### Controls.js unit tests - Evidence

| **Test** | **Evidence** | **Comment** |
| --- | --- | --- |
| 1 | N/A | Observed console and HTML to view changes |
| 2 | Z:\Alex On My Mac\Dropbox\Screenshots\Screenshot 2014-03-21 19.06.43.png |  |
| 3 | Z:\Alex On My Mac\Dropbox\Screenshots\Screenshot 2014-03-21 19.09.05.png |  |
| 4 | Z:\Alex On My Mac\Dropbox\Screenshots\Screenshot 2014-03-21 19.09.58.png |  |
| 5 | Z:\Alex On My Mac\Dropbox\Screenshots\Screenshot 2014-03-21 19.10.26.png |  |
| 6 | N/A | No error message was received |
| 7 | N/A | No warning message was received |
| 8 | N/A | No warning message was received |
| 9 | N/A | No warning message was received |
| 10 | <input class="config" type="range" id="char-speed" name="speed" onchange="updateValue(this, this.value);" min="0" max="0.2" step="0.01"> | This is the HTML generated by the function |
| 11 | <input class="config" type="range" id="char-speed" name="speed" onchange="updateValue(this, this.value);" disabled="disabled" min="0" max="0.2" step="0.01"> |
| 12 | <tr class="config" title="The speed that an ant can move"><td class="config">Speed</td><td class="config" id="input-container"><input class="config" type="range" id="char-speed" name="speed" onchange="updateValue(this, this.value);" min="0" max="0.2" step="0.01"></td><td class="config" id="char-speed-value">0.04</td><n/tr> |
| 13 | <table class="species 1"><tr class="species 1" id="1-label-row" style="color: rgb(28, 28, 28);"><td class="species 1" onclick="select(this)">Species: 1</td><td class="species 1 toggleVisibility" onclick="toggleClassVisibility(this)">-</td></tr><tr class="species 1" style="display: table-row;"><td class="species 1">Colour</td><td class="species 1" id="1-colour-data" style="background-color: rgb(28, 28, 28);"></td></tr><tr class="species 1" style="display: table-row;"><td class="species 1">Number of ants</td><td class="species 1" id="1-antNum-data">1</td></tr><tr class="species 1" style="display: table-row;"><td class="species 1">Number of Nests</td><td class="species 1" id="1-nestNum-data">0</td></tr><tr class="species 1" style="display: table-row;"><td class="species 1">Amount of food</td><td class="species 1" id="1-foodAmount-data">0.00</td></tr></table>  Z:\Alex On My Mac\Dropbox\Screenshots\Screenshot 2014-03-21 21.13.57.png |

### map.js – unit tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Action** | **Tests** | **Reason** | **Input** | **Expected** | **Result** |
| 1 | drawGrid | Tests that the grid is correctly drawn | **Typical** | GRID\_SIZE = {width: 10, height: 10}  CELL\_SIZE = {width: 10, height : 10} | A 10x10 grid | PASS |

#### map.js unit tests - Evidence

|  |  |  |
| --- | --- | --- |
| **Test** | **Evidence** | **Comment** |
| 1 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\Food\Food.draw - drawing piece of food correctly 3.png |  |

### Compatibility tests

The tool has been tested on multiple web browsers on multiple operating systems to check its compatibility. This is necessary as the tool may be set for homework and the pupils need to be able to use it at home.

*Note*: The canvas HTML 5 element which is used to display the simulation is the same in all browsers which support it for its main features. The use of the canvas element in the simulation is specifically designed to only use the core features of the canvas so that support can be guaranteed by browsers which have an implementation of the canvas element.

| **Test** | **Description** | **Reason** | **Evidence** | **Result** |
| --- | --- | --- | --- | --- |
| 1 | Chrome – Windows 8 | Chrome being the most popular browser and windows 8 being a popular operating systems | C:\Dropbox\projects\Ant-Simulation\app\tests\Browser compatibility\win8_chrome_27.0.png | PASS |
| 2 | Firefox – Windows 8 | Firefox is also a popular web browser and windows 8 being a popular operating systems | C:\Dropbox\projects\Ant-Simulation\app\tests\Browser compatibility\win8_firefox_20.0.png | PASS – The slider input html element is not supported in Firefox and so the inputs are text boxes rather than sliders |
| 3 | Safari – IOS (iPad) | Safari is the default browser on IOS and so properly most used. The iPad is one of the only portable devices which the simulation can run on smoothly, phones are not powerful enough | C:\Dropbox\projects\Ant-Simulation\app\tests\Browser compatibility\ios_iPad-3rd_5.1_portrait.jpg | PASS |
| 4 | Chrome - Ubuntu | Chrome being the most popular browser and Ubuntu being the most popular Linux distribution this test is likely to catch the small number of people who use a Linux OS | C:\Users\0x\Downloads\140203-162615-begly.github.io-9871292\140203-165738-chrome-32.0.1700.102-ubuntu-12.04-lts-41d660b1228e545dbe8d57c4dfd6b8d1.png | PASS |
| 5 | Safari – OS X | Safari is the default browser on OS X and therefore probably one of the most popular | C:\Users\0x\Downloads\140203-162615-begly.github.io-9871292\140203-165115-safari-5.1.7-mac-os-x-10.7-8c5186b2a1e265bb6967357aab5963fc.png | PASS |

### Stress tests

The simulation went through a series of stress tests to see how well it copes with large numbers of ants. Ants where picked as the independent variable as the number of ants is the primary contribution to the time complexity of the algorithm. Only worker ants are used as they act as a good average complexity (as queen ants require less processing then worker ants and soldier ants require more processing then worker ants).

#### Setup

The simulation was set with a GRID\_SIZE of 250x250. The simulation was slightly modified so that an exact number of ants would be created at random positions at the start of the simulation rather than having to wait for multiple nests to produce enough ants.

The simulation was run on chrome 32 on the windows 7 operating system. The hardware was a school computer similar to the ones used in the biology department and thus a good representation of the performance to be expected when the simulation is used.

#### Frame rate and usability

The frame rate in the simulation will not be noticeable until it gets extremely low e.g. < 2 fps. This is because the frame rate is the same as the tick time, thus meaning higher frame rates will mean the simulation runs faster and lower frame rates means the simulation runs slower. This is done to increase performance of the simulation so that it can run as fast as possible so the user can see how the change of characteristics affect the ant’s behaviour faster.

#### Results

| **Test** | **Tests** | **Reason** | **Expected** | **Result** |
| --- | --- | --- | --- | --- |
| 1 | 1 ant | The starting number of ants on the simulation. This is certain to happen in every simulation | Very smooth frame rate (> 60fps) | PASS |
| 2 | 10 ants | A small nest of ants (< 20 ants per nest). This will very nearly happen every simulation run | Very smooth frame rate (> 60fps) | PASS |
| 3 | 100 ants | Multiple medially sized nests (> 20 ants per nest). This will happen in simulations left on few about 2 minutes | Smooth frame rate (> 30fps) | PASS |
| 4 | 1000 ants | Multiple large nests (> 80 ants per nest). This is not likely to happen to most users | Useable frame rate (> 5fps) | PASS |
| 5 | 10000 ants | Multiple very large nests (>200 ants per nest). This may not happen atoll in most simulations, no matter how long they are left on for due to the characteristics chosen and either ants dying out due to being out competed for food or lack of food | Very choppy frame rate (> 0.25fps) | PASS |
| 6 | 100000 ants | Many multiples of extremely large colonies of ants (>5000 ants per nest). This is extremely unlikely to occur as it would require a significant amount of time to get the simulation to this point as well as an extremely large map and large food source | Unusable (< 0.25fps) | PASS |

#### Stress tests – Evidence

| **Test** | **Evidence** | **Comment** |
| --- | --- | --- |
| 1 | C:\Dropbox\projects\Ant-Simulation\app\tests\stress\1 ant.PNG | 200 fps |
| 2 | C:\Dropbox\projects\Ant-Simulation\app\tests\stress\10 ants.PNG | 100 fps |
| 3 | C:\Dropbox\projects\Ant-Simulation\app\tests\stress\100 ants.PNG | 40 fps |
| 4 | C:\Dropbox\projects\Ant-Simulation\app\tests\stress\1000 ants.PNG  Test - 1000 ants (8 fps) | 8 fps |
| 5 | C:\Dropbox\projects\Ant-Simulation\app\tests\stress\10000 ants.PNG  Test - 10000 ants (1 fps) | 1 fps |
| 6 | C:\Dropbox\projects\Ant-Simulation\app\tests\stress\100000 ants.PNG | N/A |

### User interface functional testing

| **Test** | **Tests** | **Method** | **Expected** | **Result** |
| --- | --- | --- | --- | --- |
| 1 | The run button starts and stops the simulation. | Press the pause button and then press the run button | When pause button pressed simulation stops and the pause button will change to a run button. When the run button is pressed the simulation starts from the place it left off and the run button changes back into a pause button | PASS |
| 2 | Press the space bar | PASS |
| 3 | The step button steps the simulation forwarded a single tick. | Press the step button | All ants to go through a single update cycle i.e. do a single action such as pick up a piece of food or attack an ant. All ants (apart from those collecting food) will move an amount equal to their species speed characteristic | PASS |
| 4 | Press the “s” key | PASS |
| 5 | The restart button restarts the simulation. | Press the restart button | The simulation restarts, keeping all characteristics for the starting species the same | PASS |
| 6 | Press the “r” key | PASS |
| 7 | Species data selection. | Click on the species from the species panel. (Selecting species 46) | Map to centre on the first nest in the spices. The selected species to be highlighted in the species data panel. The characteristics of the selected species to appear in the characteristics panel | PASS |
| 8 | Expanding species data | Press the + button of species 87 | Data about the species expands showing the species colour, number of ants, number of nests and amount of food | PASS |
| 9 | Contracting species data | Press the – button of species 87 | Data about the species colour, number of ants, number of nests and amount of food to be hidden | PASS |
| 10 | Pan around simulation | Click and drag mouse on simulation | Simulation to move under the mouse until mouse is released | PASS |
| 11 | Press arrow keys | PASS |
| 12 | Zoom into simulation | Scroll mouse wheel | Simulation to either zoom or zoom out depending on direction of scroll/+ or – key | PASS |
| 13 | Press + and – keys | PASS |
| 14 | The random button randomizes the selected species characteristics | Pressing the random button | The characteristics of the selected species to be changed to random values. The update button to highlight. And the ants health to change | PASS |
| 15 | The reset button resets selected species characteristics to default values | Press the reset button | The characteristics of the selected species to change to the default values. The update button to highlight. And the ants health to change | PASS |
| 16 | The update button pushes the changed value to the simulation | Press the update button | The update button to return to its normal colour. The species in the simulation value to be updated | PASS |

#### User Interface functionality tests – Evidence

|  |  |  |
| --- | --- | --- |
| **Test** | **Evidence** | **Comment** |
| 1 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\run button\pause pressed.PNGZ:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\run button\run pressed.PNG | The image on the right is when the pause button is pressed. The image on the left is after the button has been pressed again |
| 2 |

| **Test** | **Evidence** | **Comment** |
| --- | --- | --- |
| 3 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\step button\step pressed when paused.PNGZ:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\step button\paused simulation.PNG | The first image is before the step button is pressed. The image on the left is after the step button has been pressed |
| 4 |

|  |  |  |
| --- | --- | --- |
| 5 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\restart button\pre restart 2.PNGZ:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\restart button\restart.PNG | The image on the left is before the restart. The image on the left is after the restart |
| 6 |
| 7 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\species selection\species 1 selected.PNG | The first image is species 1 is selected. The second image is species 46 selected. Notice that the exploitative influence changes. And that the simulation jumps to a nest of that species |
| 7 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\species selection\species 46 selected.PNG |
| 8 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\species data expansion\1 contracted.PNG |  |
| 9 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\species data expansion\unexpanded.PNG |  |

|  |  |  |
| --- | --- | --- |
| 10 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\pan around map\2 - Click and drag.pngZ:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\pan around map\1 - Place cursor over map.png | The first image shows the map before the pan and the second image shows the map after the pan. Notice that the food moves and so do the purple ants |
| 11 |

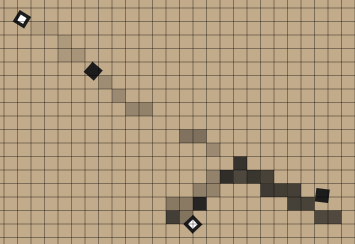
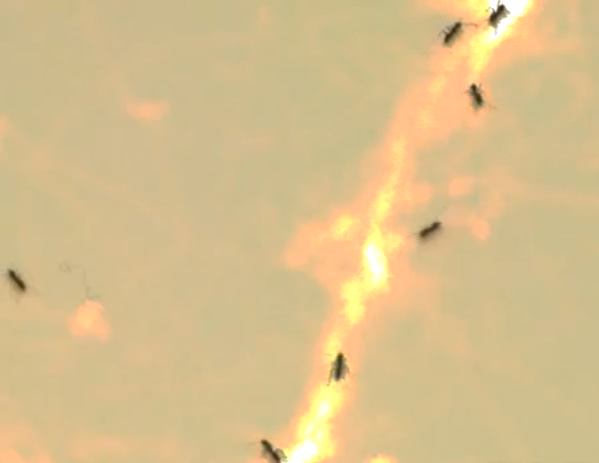
|  |  |  |
| --- | --- | --- |
| 12 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\zoom in map\1 - Place cursor over map.pngZ:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\zoom in map\2 - Scroll.png | The first image is before the zoom and the second image is after the zoom |
| 13 |
| 14 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\random button\pre random.PNGZ:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\random button\post random.PNG | The first image is before the random button is pressed. The second image is after the random button has been pressed. Notice that the update button is also heightened |
| 15 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\reset button\pre reset.PNGZ:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\reset button\post reset.PNG | The image on the right is before the reset. The image on the left is after the reset |
| 16 | Z:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\update button\post update.PNGZ:\Alex On My Mac\Dropbox\projects\Ant-Simulation\app\tests\inputs\update button\update pressed.PNG | The image on the right shows a sample of the panel before the update and the image on the left is after the update |

### Biological tests

The simulation is based on ants, the behaviour of the simulated ants is changed by altering a species characteristics. This behaviour can sometimes closely resemble real ants however it can be equally a long way of how real ant behaves. For example by changing the antenna size characteristic to 0 will mean that ants can no longer smell pheromones. The behaviour of ants in this state will not model the behaviour of real ants (which can smell pheromones). This means that in order to test the biological trueness of the simulation would be to find the characteristics in the simulation which most closely match a real ant; this is very difficult as the simulation only has a limited range of characteristics to choose from while real ants have a much more complex system of genes.

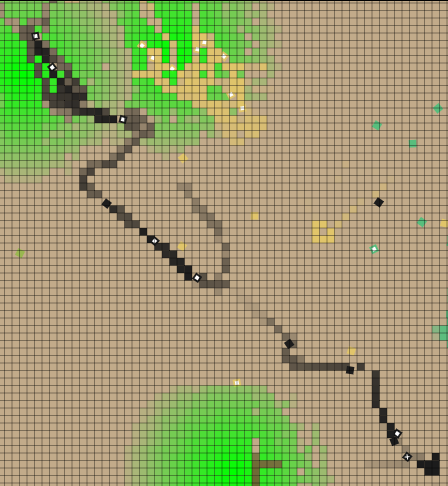
A comparison between real ants and the simulated ants will be used to compare how well the simulated ants model the real ants. Also approximations of behaviour will also be compared between the two.

#### Pheromones

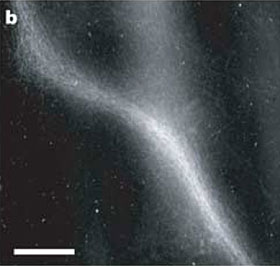
Pheromones laid by real ants are most concentrated where most ants have been as they cumulate. This is seen in the picture of real ants by the brighter yellow colour. This can be seen in the simulation by a darker colour pheromone. In videos, ants will tend to head in the direction which has the strongest concentration of pheromones. Again this is mirrored in the simulation (See tests 17, 18 and 19 in Ant.js unit tests). Real ants will also sometimes leave the trial to go off looking in a new direction. This is shown in the diagram of the simulation where the top right ants did not follow the stronger pheromone trial left by the ant in the centre left. And in the picture of real ants by the ant on the far left of the screen who left the trial in search for food. Finally pheromones in real life evaporate over time, this is also modelled in the simulation as pheromones concentration decrease as time increases.

#### Searching for food

Real ants walk randomly until an ant finds food, it then navigates back to the nest (depending on type of ant either following a pheromone, memory or randomly). Other ants then follow the trial left by the first ant to find the food, reinforcing the strength of the trial making it more popular. This can all also be seen in the simulated ants.

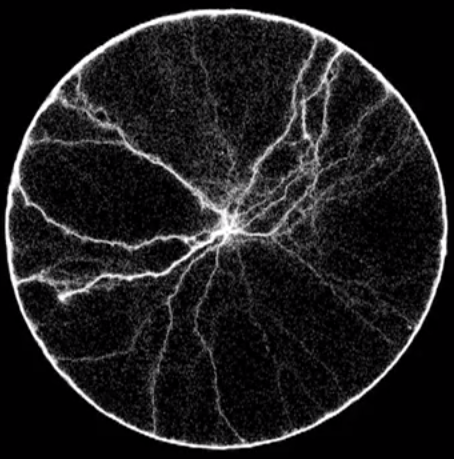
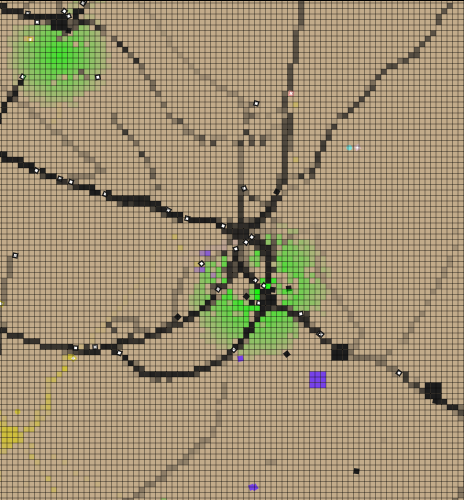
The screen shot shows a trial which leads from the nest to food. A similar looking trial is seen in the picture of real ants leading from the nest to food.

NEST



FOOD

#### Trail patterns

Real ant trials come into the nest from all directions, the main trials (strongest concentration pheromones) are roughly 70 degrees apart (to maximise spread to new food sources). This is very similar to the simulation which has trials from all directions and the main trials are separated by about 70 degrees also. The concentration of pheromones is also similar, although the scale of the simulation and the real pheromone trials of ants are different the pattern of most concentrated pheromones around the nest with roughly straight thin lines of high concentration pheromones leading off in multiple directions as well as a small number of considerable weaker pheromones from all directions can be seen in the real pheromones and the simulated.

# Section 5: System Maintenance

## System overview

The system allows a user to see how the behaviour of ants is determined by their characteristics as well as allowing the editing of these characteristics to design specific species. The system is a single page web application which uses the canvas (introduced in the HTML5 specification) for rendering all shapes. The simulation implements three different types of ants:

* **Worker ants** – these ants find and collect food to feed their nest.
* **Soldier ants** – these ants guard their nests as well as the worker ants and food sources. They are also responsible for attacking other species of ants.
* **Queen ants** – these are responsible creating new nests.

The simulation also implements a simple ant nest which allows the creation of new ants and acts as a store for deposited food. The simulation includes food, used by ants to survive and in particular used by workers who collect it and deposit it at the nest. Food can re-grow over time. Pheromones are also implemented in the form of worker ant food trials.

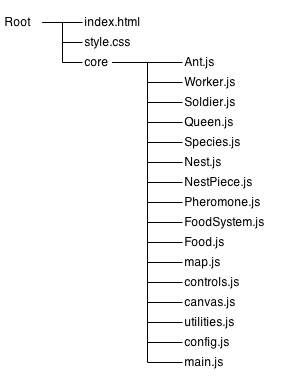
In the system the user has the ability to view how ants behave and evolve over time. The user can also change characteristics of ants to create a user-defined species and can move around the simulation, zoom in and out and reset the simulation.

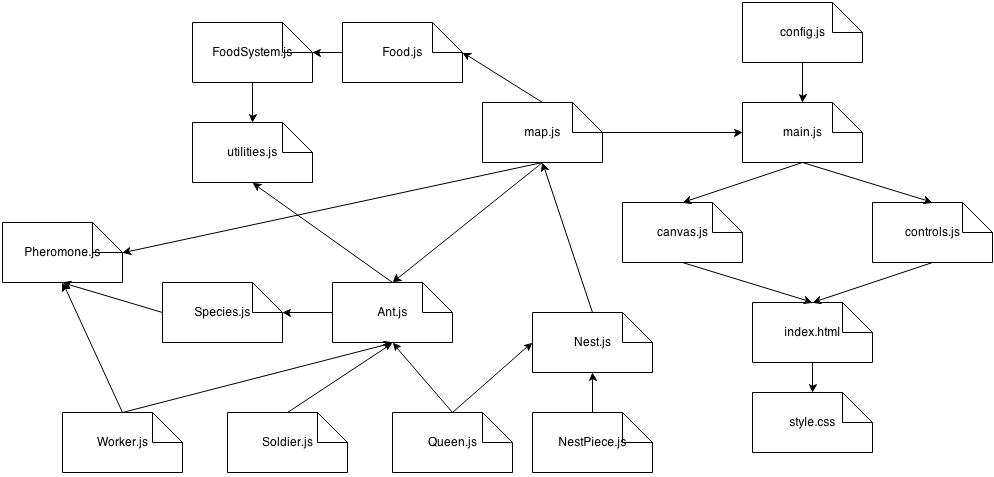
## Modular structure overview

The applications files are split up into logical groups of code. Each group controls a single part of the system.

### File structure

The following diagrams describe how the files are organised in the implementation. The first diagram shows the overall file directory structure. The second diagram shows how files relate to one another. A link shows that one file directly refers to code in another file. This is helpful to know when modifying the code as it shows which files may be affected directly by the change. Notice that main.js is the primary link between the JavaScript and the interface and thus changes to the interface may affect main.js.

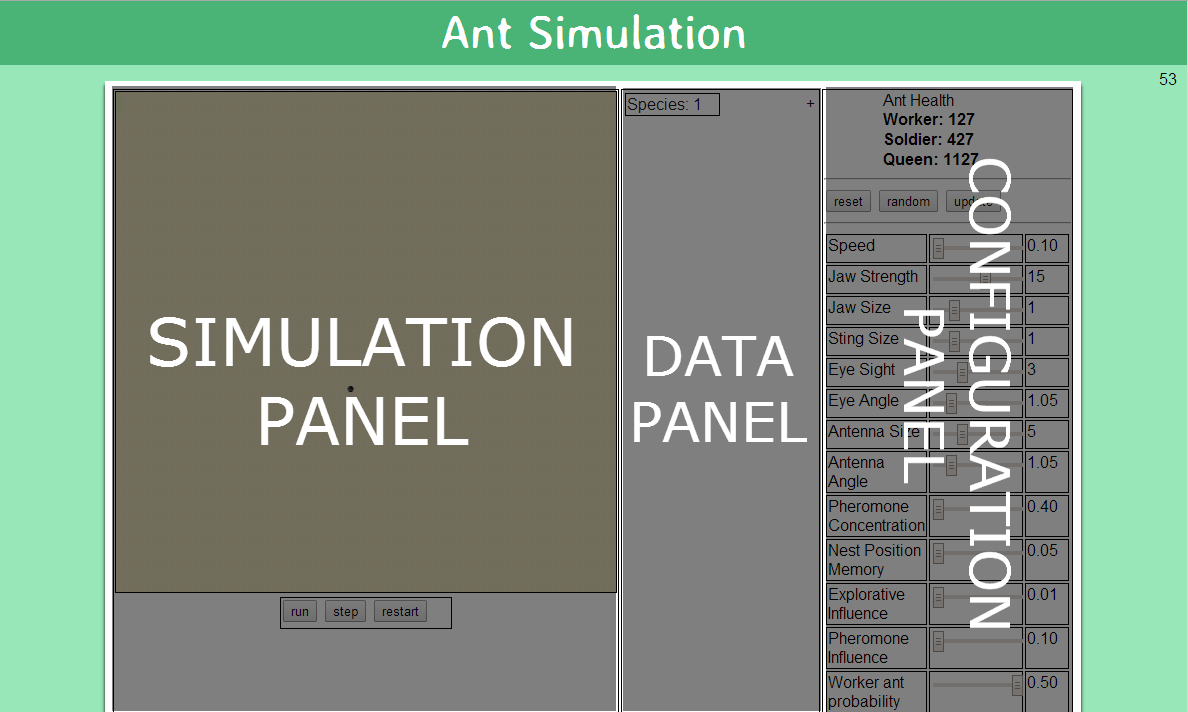




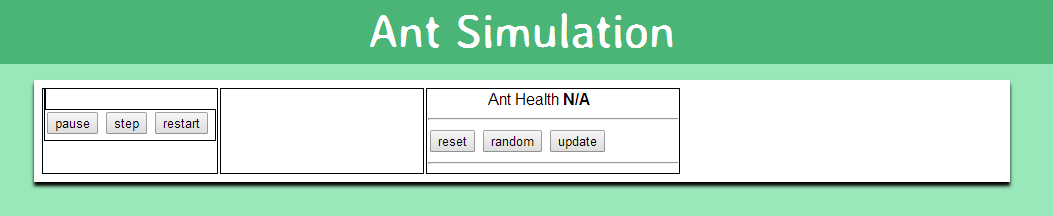
### Index.html - website structure

The application is a single page website. The page is arranged into three sections:

* The simulation panel – contains the simulation itself as well as controls for the simulation.
* The data panel – contains information about the different species in the simulation, such as how many ants the selected species currently has.
* The configuration panel – contains inputs which allow the user to change the values of characteristics for the currently selected species.



The page is written in HTML. The configuration panel is completely populated at run time with characteristics. This is done to more easily allow characteristics to be added to the system as they are only required to be added to core/config.js. The same is true for the data panel. Species data is populated throughout the simulation as species change. This means that the HTML code in index.html is only a small part of the actual HTML displayed in the system. The following is the page with scripts turned off, to demonstrate how much of the page is generated dynamically at run time.



### Style.css - page styling

The page is styled in CSS *Note*: This styling only affects the HTML generated in the page. The simulation itself uses the HTML canvas API and is styled separately see configuration for styling the canvas.

### Core system

The application is split into 10 classes and 5 modules. Each is purposed to represent or contain functions applicable to a certain aspect of the simulation.

Ants.js – the Ant class

The Ant class is the parent class of all other ant classes in the system (worker, soldier and queen). It defines the basic properties of an ant (size, position, id, species, home nest…) and its basic actions (takeFood, getFood, scan, smell…).

Worker.js – the Worker class

The Worker class represents a worker ant. It defines properties and methods to do with the search, collection and deposition of food.

Soldier.js – the Soldier class

The Soldier class represents a soldier ant. It defines properties and methods to do with the guarding and attacking of other ants and nests within the Map.

Queen.js – the Queen class

The Queen class represents a queen ant. It defines properties and methods to do with the location and creation of a new nest.

Species.js – the Species class

The Species class represents a species in the simulation. A species is a collection of specific characteristic values. The species class defines properties and methods to do with the storing and mutation of characteristics. All ants and nests have a species. Ants born from a particular nest will inherit the species of the nest. However if the ant is a queen ant, there is a chance that the species will be mutated and the queen will inherit the mutated species.

If a species is mutated, it means that a single characteristics value has been altered from its original value.

In the simulation, each species has a unique colour. The colour is randomly generated when the species is created. All ants and nests belonging to the species will also appear to be this colour. This allows entities from the same species to be easily identified in the simulations interface.

For data collection purposes, the species class keeps track of all ants and nests which belong it its species. This is used in the data panel where a species data is shown.

Nest.js – the Nest class

The Nest class represents an ant’s nest. It defines properties and methods to do with the creation of new ants.

NestPiece.js – the NestPiece class

The NestPiece class represents a piece of the nest on the map. The nest which appears on the map is made up of a number of nest pieces, one per cell. This is done so that when a nest is attacked, parts of the nest can be destroyed while the rest of the nest remains. The nest pieces do not control any actions to do with the running of the nest and are simply physical representations of a nest object on the map. The Nest class on the other hand is the entity which controls the nest. Once all of a nests nest pieces are destroyed the nest will be destroyed. *Note*: See Nest vs. NestPieces in Difficult to Understand Code section for more details on their differences.

Pheromone.js – the Pheromone class

The Pheromone class represents a pheromone on the map. It defines methods and properties to do with adding, removing and updating a pheromone. A pheromone has a concentration this will slowly decay as it evaporates over time. When a pheromones concentration is less than or equal to 0 the pheromone is removed from the map.

FoodSystem.js – the FoodSystem class

The FoodSystem class defines properties and methods to place and grow food in the map. *Note*: The amount of food placed in the map is controlled in the config.js file. See the Environment section in configuration for details on how to edit the amount of food or how it grows.

Food.js – the Food class

The Food class represents a single piece of food. It defines properties and methods to place, draw and grow food. A piece of food contains an amount property; this is the amount of food that the piece contains. Different pieces of food can contain different amounts/concentrations of food; this is controlled by the FoodSystem object which created the food. Foods of different concentrations appear different shades of the FOOD\_COLOUR. A darker shade means the food piece contains a higher concentration of food. While a lighter colour contains a lower concentration of food.

map.js – the map module

The map module defines functions relating to the creation, drawing and navigation of the map. The MAPitself is a global variable which represents the map which entitles are placed in the simulation. The map is comprised of cells; each cell can contain multiple entities:

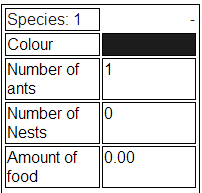
* MAP[index].ants – is an array of all the ants in the current cell.
* MAP[index].pheromones – is an array of all the pheromones in the cell.
* MAP[index].food – contains either void(0) if there is no food in the cell or a food object if there is. *Note*: unlike MAP[index].ants and MAP[index].pheromones, MAP[index].food is not an array i.e. can contain only a single object.

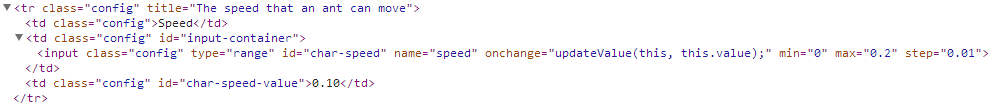
*Note*: index in the above list is an integer relating to the index of a cell in the MAP array this varies from 0 up to the GRID\_SIZE.width \* GRID\_SIZE.height i.e. the NUMBER\_OF\_CELLS constant.

*Note*: Ants can move distances smaller than the size of a single cell, this means that ants may exist between cells when drawn. However on the MAP the cell which the ant lies in mostly will be used as the cell which the ant is in i.e. the cell which the majority of the ants body lies in will be used as an approximation to its position.

controls.js – the controls module

The controls module is responsible for HTML generation. It controls functions for the creation and removal of HTML elements. The module is used when the page is first loaded to resize the canvas and generate the inputs in the configuration panel from the characteristics defined in config.js. It is then used to generate species data when new species are created, and update species data each tick.

The following are examples of some of the HTML generated by the controls module:

C:\Dropbox\projects\Ant-Simulation\writeup\assests\Maintainance\HTML generation\Generated HTML configuration panel - referes to.PNG

### canvas.js – the canvas module

The canvas module contains a collection of functions which operate on the canvas as well as use the canvas API. The canvas is a new tag in the HTML5 specification, it allows for dynamic, scriptable rendering of 2D shapes. Thecanvas module impended mainly contains shortcuts for drawing shapes such as rectangles, lines, arcs and circles. However also contains functions for resizing and clearing the canvas.

*Note*: The canvas module is the only location where calls to the canvas API are used. This is done so that if the specification changes or is updated; only a single module must be changed to reflect the changes in the API. This makes the project much easier to maintain.

### utilities.js – the utilities module

The utilities module contains a collection of functions which are used by multiple classes and modules. It contains functions for generation of random values, calculation of distances, angles and boundaries, coordinate manipulating and conversion, retrieving selections of cells from the MAP and creation of ants.

### config.js – the configuration file

The configuration file contains all of the system configurable values. Configuration for the entire system is done using this file. This is to improve maintainability as it means only a single file must be changed and it also means it is easier to locate configuration options.

### main.js – the main program

This file is contains the main update loop for the simulation, and creates the simulation environment as well as defying event listeners for events such as shortcut key presses. The main update loop loops through every cell in the map and updates every pheromone; piece of food and ant each tick. The environment created creates a FoodSystem, generates food and adds single queen ant to start the simulation.

### Tests

The system contains a large number of unit tests for all parts of the system. See Test Driven Development section for more details.

## Classes

*Note*: parameters of the function are appended with **[param].**

|  |  |
| --- | --- |
| Type | Description |
| integer | A whole number |
| number | A real number |
| string | A list of characters |
| Boolean | A true/1 or false/0 |
| \* object | An object |

*Note*: As JavaScript is weakly typed the types stated are the intended types / how they are used in the system.

*Note*: In JavaScript objects are associative arrays, augmented with prototypes.

### Ant

| **Function name** | Variables | Type | Description |
| --- | --- | --- | --- |
| Ant | this.size | width : integer, height : integer | The size of the ant in pixels (default: CELL\_SIZE) |
| this.coord | x : number, y : number | The coordinate of the ant |
| this.id | integer | The unique ant id |
| this.species | Species object | The ant’s species which determines its characteristics |
| this.type | integer | The type of ant, use the ANT\_TYPE object when setting. e.g. ANT\_TYPE.worker |
| this.nest | Nest object | The ant’s home nest where it was born |
| this.colour | string | The hexadecimal colour of the ant e.g. ‘#FF0000’ (default: “#1C1C1C”) |
| this.health | number | The health of the ant, if less than or equal to 0 the ant is dead |
| this.hungerThreshold | number | The value of health bellow which the ant is determined to be hungry (default: 100) |
| this.healthRate | number | The rate at which the ant’s health decreases per tick (default: 0.1). |
| this.alive | boolean | Represents the ant’s living state (default: true). |
| this.goal | integer | The current goal which the ant is trying to accomplish, use the GOAL object when setting e.g. GOAL.findFood (default: GOAL.none) |
| this.target | x : number, y : number | The coordinate of a target the ant has chosen, the type of target depends on the ant i.e. a worker ant targets food while soldier ants target enemy ants (default: void(0)) |
| this.itemsInView | ants: [Ant object], food : [Food object] | Holds arrays of all ants and food within view (default: ants: [], food: []) |
| this.pheromonesInRange | [Pheromone object] | An array of all the pheromones in the ant’s antenna range (default: []) |
| this.sleep | integer | The number of ticks the ant needs to sleep for. Used for tasks which require actions which take multiple ticks to complete (default: 0) |
| this.followingPheromone | boolean | Used to tell if an ant is following a pheromone (default: false) |
| this.direction | number | The angle which the ant is facing in radians from the vertical axis clockwise (default: \*random direction\*) |
| this.prioritizeDirection | number | The direction the ant will tend to move in, used to achieve straighter more realistic paths (default: \*random direction\*) |
| Ant.addToMap | N/A |  | Adds the current position of the ant on the map |
| Ant.removeFromMap | N/A |  | Removes the current position of the ant from the map |
| Ant.isHungry | N/A |  | Determines whether the ant is hungry or not  Returnboolean - true if the ant is hungry else false |
| Ant.updateSleep | N/A |  | Updates this.sleep variable to simulate time passing |
| Ant.isFood | [param] food | Food object | Determines whether a piece of food exists or not  Returnboolean - true if piece of food exists else false |
| Ant.takeFood | [param] food | Food object | Takes a single piece of food  Returnboolean - true if there is still food left, else false |
| Ant.atNest | N/A |  | Determines whether the ant is currently at its own nest i.e. standing on top of a nest piece entity  Returnboolean - true if the ant is standing on its nest, else false |
| Ant.seeNest | N/A |  | Determines whether the ant can see its own nest  Returnboolean - true if ant can see its nest, else false |
| Ant.findFoodTarget | N/A |  | Choose the target piece of food the ant should go for |
| Ant.getFood | N/A |  | Walk towards food until on top of it and then pick it up one piece at a time |
| Ant.useFood | N/A |  | Determines the best use of food i.e. either eating the food if hungry or carrying it |
| Ant.scan | N/A |  | Looks at all blocks in front of the ant within eyesight, places all items of interest into this.itemsInView |
| Ant.smell | N/A |  | Similar to this.scan - Looks at all blocks in front of the ant within antenna size range and places all pheromones into this.pheromoensInRange |
| Ant.secrete | N/A |  | Secrete pheromones onto the map |
| Ant.wonder | N/A |  | Wonder around the map, following pheromones of own species otherwise picking random direction to move in |
| Ant.move | N/A |  | Updates the ants coordinates |
| Ant.die | N/A |  | Removes the ant from the simulation |

### Worker

**Inherits Ant**

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| Worker | this.coord | x : number, y : number | The coordinate of the ant. |
| this.id | integer | The unique ant id |
| this.type | integer | The type of ant, use the ANT\_TYPE object when setting. e.g. ANT\_TYPE.worker (default: ANT\_TYPE.worker) |
| this.direction | number | The angle which the ant is facing in radians from the vertical axis clockwise (default: \*random direction\*) |
| this.prioritizeDirection | number | The direction the ant will tend to move in, used to achieve straighter more realistic paths (default: \*random direction\*) |
| this.carrying | integer | The amount of food the ant is carrying (default: 0) |
| this.carryingThreshold | integer | If an ant is carrying more food then this value and cannot see any food near it, ant will return to the nest to deposit the food (default: 4) |
| Worker.canCarry | N/A |  | Determines if an ant can carry food or not  Returnboolean - true if ant can carry food else false |
| Worker.depositeFood | N/A |  | Navigate towards the nest and deposit food at the nest |
| Worker.dropFood | N/A |  | Drop a single piece of food at the nest  Returnboolean - True if there is no more food to drop off, else false |
| Worker.useFood | N/A |  | Determines the best use of food i.e. eating it if hungry or carrying it |
|  | index - The MAP index of the ant | integer |
|  | food - The food object which is being used | Food object |
| Worker.doTask | N/A |  | Performs the actions required to complete a task |
| Worker.updateGoal | N/A |  | Determines if a goal has been completed or not and updates the next goal for the ant |
| Worker.updateHealth | N/A |  | Updates the ant’sthis.health variable. Differs from ant.updateHealth as allows ants to eat food they are carrying if hungry |
| Worker.draw | [param] ctx | Canvas context object | Draws the ant onto the canvas context |
|  | scaledCoord - The coordinate of the map scaled to pixels | integer |
| Worker.update | N/A |  | Update the ant each tick |

### Queen

**Inherits Ant**

| Function name | Variables | Type | Description |
| --- | --- | --- | --- |
| Queen | this.coord | x : number, y : number | The coordinate of the ant |
| this.id | integer | The unique ant id |
| this.type | integer | The type of ant, use the ANT\_TYPE object when setting. e.g. ANT\_TYPE.worker (default: ANT\_TYPE.queen) |
| this.steps | integer | The number of steps the queen will take until reaching the nest site |
| Queen.doTask | N/A |  | Decide what actions need to be done to accomplish a task |
| Queen.updateGoal | N/A |  | Checks to see if the goal is accomplished and updates it if necessary |
| Queen.pickDirection | N/A |  | Used to pick a direction in which the Queen will walk a specific number of steps in (this.steps) and then create a nest |
| Queen.createNest | N/A |  | Creates a nest object |
|  | nest - The MAP index of the ant | Nest object |
|  | index - The index of the queen ant in this.species.ants used to determine which ant to remove from the list | integer |
| Queen.draw | [param] ctx | Canvas context object | Draw the queen onto the canvas context |
|  | scaledCoord - The scaled MAP coordinate to pixels | integer |
| Queen.update | N/A |  | Update the queen each tick |

### Soldier

**Inherits Ant**

| Function name | Variables | Type | Description |
| --- | --- | --- | --- |
| Soldier | this.coord | x : number, y : number | The coordinate of the ant |
| this.id | integer | The unique ant id |
| this.type | integer | The type of ant, use the ANT\_TYPE object when setting. e.g. ANT\_TYPE.worker (default: ANT\_TYPE.soldier) |
| this.direction | number | The direction in radians from the vertical axis clockwise (default \*random direction\*) |
| this.prioritizeDirection | number | The direction the ant will tend to move in, used to achieve straighter more realistic paths (default: \*random direction\*) |
| this.targetAnt | Ant object | The enemy ant which the soldier is targeting to attack (default: void(0)) |
| this.moving | boolean | Used for determining when an ant is in a static position e.g. guarding the nest (default: false) |
| this.steps | integer | Used for moving an ant a certain number of steps away from an object |
| this.nearNest | boolean | Determine if an ant is close to the nest (even if it is not in view) (default: false) |
| this.nearFood | boolean | Determine if an ant is close to food (even if it is not in view) (default: false) |
| Soldier.soldiersInView | N/A |  | Determines if there are other friendly soldiers in view  Returnboolean – true if there are soldiers in view else false |
| Soldier.seeFood | N/A |  | Determines if an ant can see food  Returnboolean – true if the ant can see food else false |
| Soldier.pickTarget | N/A |  | Chooses which ant the soldier should target |
| Soldier.follow | N/A |  | Sets the ants direction to intercept the path of the targeted ant so that it can attack |
| Soldier.attack | N/A |  | Attack the targeted ant if in range |
|  | dist - The distance between the ant and its target. | number |
| Soldier.updateHealth | N/A |  | Updates the soldier ant’s health differs from Ant.updateHealth as will change soldier’s goal if hungry |
| Soldier.updateSteps | N/A |  | Updates the this.steps variable |
| Soldier.guardNest | N/A |  | Controls soldier’s logic if assigned goal of guarding the nest |
| Soldier.guardPheromone | N/A |  | Controls soldier’s logic if assigned goal of guarding pheromone trials |
| Soldier.guardFood | N/A |  | Controls soldier’s logic if assigned goal of guarding food |
| Soldier.findFood | N/A |  | Controls soldier’s logic if assigned goal of finding food e.g. if the ant is hungry |
| Soldier.doTask | N/A |  | Performs the actions required to complete a task |
| Soldier.updateGoal | N/A |  | Determines if a goal has been completed or not and updates the next goal for the ant |
| Soldier.draw | [param] ctx | Canvas context | Draw the ant onto the canvas context |
|  |  | object |  |
| scaledCoord - The scaled MAP coordinate to pixels | integer |  |
| Soldier.update | N/A |  | Update the ant each tick |

### Species

| Function name | Variables | Type | Description |
| --- | --- | --- | --- |
| Species | this.id | integer | The unique ant id |
| this.ants | [Ant object] | Array of all ants which belong to the species (default: []) |
| this.nests | [Nest object] | Array of all nests which belong to the species (default: []) |
| this.chars.speed | number | The speed the ant moves, 1 = 1 cell per tick (default: 0.25) |
| this.chars.antennaSize | integer | The range the ant can smell pheromones (default: 5) |
| this.chars.jawStrength | integer | The strength of the ants jaw (determines how much food the ant can carry) (default: 10) |
| this.chars.jawSize | integer | The amount of damage a soldier ant does when attacking (default: 1) |
| this.chars.stingSize | integer | The range which the solider ant can attack (default: 1) |
| this.chars.eyeSight | integer | The range the ant can see items In front of it (in number of cells) (default: 5) |
| this.chars.eyeAngle | number | The angle of the sector the ant can see in front of it (default: π/2) |
| this.chars.antennaAngle | number | The angle of the sector the ant can smell pheromones in front of it (default: π/2) |
| this.chars.pheromoneConcentration | number | The concentration of pheromones secreted (default: 0.4) |
| this.chars.nestCoordMemory | number | A measure of how well the ant knows where the nest is, used when navigating to the nest, represents memory of familiar landmarks near the nest (default: 0.1) |
| this.chars.explorativeInfluence | number | The likelihood of an ant changing direction rather than continuing going the direction it’s facing (default: 0.05) |
| this.chars.pheromoneInfluence | number | How likely it is that an ant will follow a pheromone (default: 0.9) |
| this.chars.queenStepsMin, this.chars.queenStepsMax | number | The range of steps the queen will take when navigating to a new nest site i.e. lower values mean closer nests (default: min: 200, max: 800) |
| this.chars.reproduceWorkerProb, this.chars.reproduceQueenProb, this.chars.reproduceSoldierProb | number | The probability a particular type of ant will be born compared with others (default: worker: 0.5, queen: 0.05, soldier: 0.1) *Note*: gets normalized so does not need to add to one |
| this.chars.reproduceWorkerFoodCost, this.chars.reproduceQueenFoodCost, this.chars.reproduceSoldierFoodCost | number | The amount of food required to create this type of ant (this is the amount of food the ant will start with when born) (default: worker: 5, queen: 25, solider: 8) |
| this.chars.reprodcutionRate | number | The chance each tick of creating a new ant (default: 0.05) |
| this.colour.worker, this.colour.soldier, this.colour.queen, this.colour.nest, this.colour.pheromone | string | The hexadecimal colours given to each type of ant/nest/pheromone. *Note*: often all values are the same (default: ‘#1C1C1C’ for all) |
| this.murationRate | number | The chance that a characteristic will be mutated |
| Species.mutateChar | [param] characteristic | string | Mutate a specific characteristic  Return number – The mutated value of the characteristic |
| Species.mutate | N/A |  | Mutate a single characteristic in the species  Return Species object – The new mutated species |
| altChars - The current characteristics of the species. | object |
| altCharacterisitc - The random characteristic which will be mutated. | string |
| altValue - The new value of the altCharacterisitc. | Variable depending on characteristic being mutated. |
| species - The species new species. | Species object |
| Species.createSpecies | [param] chars | object | The characteristic set of the new species  Return Species object – The new species |
| species - The new species | Species object |
| colour - The colour of the new species | string |

### Nest

| Function name | Variables | Type | Description |
| --- | --- | --- | --- |
| Nest | this.nestSize | x: integer, y: integer | The number of pieces the nest extends in both the x and the y directions (default: NEST\_SIZE) |
| this.coord | x: number, y: number | The coordinate of the nest |
| this.id | integer | A unique identifier |
| this.species | Species object | The species which the nest belongs to |
| this.pieces | [NestPiece object] | An array of the NestPiece objects belonging to the nest (default: []) |
| this.health | number | The health the nest has (default: 10000) |
| this.hungerThreshold | number | The threshold below which the nest is hungry and tries to preserve food (default: 1000) |
| this.healthRate | number | The rate at which the nests health reduces each tick (default: 0.1) |
| this.alive | boolean | If nest is alive or not, needed if nest dies mid execution so does not keep acting as if it is alive for the rest of the tick (default: true) |
| Nest.addNestPiece | [param]coord | x: number, y: number | Create a single nest piece |
| Nest.createNest | N/A |  | Creates all the nest pieces |
| Nest.die | N/A |  | Kills the nest so that it is no longer updated and is removed from the map |
| Nest.getCost | [param]type – The ant type. | integer – use the ANT\_TYPE object when defining. | Returns the cost in the amount of health needed to create a specific type of ant  Return number – the health required |
| Nest.calcSpeciesCost | N/A |  | Calculates the cost of the characteristics due the spices  Return number – the health cost |
| Nest.viable | [param]type – The ant type. | integer – use the ANT\_TYPE object when defining | Determine whether or not it is viable to create a specific type of ant  Returnboolean – true if it is viable else false |
| Nest.createAnt | [param] type – The ant type. | integer – use the ANT\_TYPE object when defining. | Create a new ant |
| cost - The health cost of creating the type of ant | number |
| Nest.reproduce | N/A |  | Determines what type of ant to create |
| prob - A random number which will be used to determine what type of ant is created | number |
| chars - The characteristics of the species | number |
| sum - The sum of ant probabilities | number |
| queenProb, soldierProb, workerProb - The respective normalized probabilities of each type of ant | number |
| ordered - The ordered normalized probabilities of each type of ant being created | [{prob : number, type: integer}] |
| Nest.updateHealth | N/A |  | Update the nests health |
| Nest.update | N/A |  | Updates the nest each tick |

### NestPiece

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| NestPiece | this.size | width: number, height: number | The size the piece will be in pixels (default: CELL\_SIZE) |
| this.coord | x: number, y: number | The coordinate of the piece |
| this.nest | Nest object | The nest object which controls the piece |
| this.id | integer | A unique identifier |
| this.type | integer – use the ANT\_TYPE object when defining | The type of object the piece is, as the nest piece is essentially treated as a static ant and is stored under ant in MAP, the type is needed so ants can identify the piece (default: ANT\_TYPE.nest) |
| this.heatlh | number | The health the piece has |
| NestPiece.addToMap | N/A |  | Adds the piece to the map |
| Nest.removeFromMap | N/A |  | Removes the piece from the map |
| Nest.die | N/A |  | Kills the nest piece stopping it from being updated and removes it from the map |
| Nest.draw | [param]ctx | Canvas context object | Draws the nest piece onto the canvas context |

### Pheromone

|  |  |  |  |
| --- | --- | --- | --- |
| Function name | Variables | Type | Description |
| Pheromone | this.concentration | number | The concentration of the pheromone |
| this.coord | x: number, y: number | The coordinate of the pheromone |
| this.size | x: integer, y: integer | The size of the pheromone in pixels (default: CELL\_SIZE) |
| this.species | Species object | The species which the pheromone belongs to |
| Pheromone.addToMap | N/A |  | Adds the pheromone to the map |
| Pheromone.removeFromMap | N/A |  | Removes the pheromone from the map |
| Pheromone.draw | [param]ctx | Canvas context object | Draw the pheromone onto the canvas context |
| Pheromone.update | N/A |  | Updates the pheromones concentration each tick |

### FoodSystem

| Function name | Variables | Type | Description |
| --- | --- | --- | --- |
| FoodSystem | this.variation | min: integer, max: integer | The minimum and maximum amounts of food a single piece can contain |
| this.colour | string | The hexadecimal colour of the piece of food (default: FOOD\_COLOUR). *Note*: opacity depends on food amount |
| FoodSystem.addFoodBlob | [param]coord – The coordinate of the food | x: number, y: number | Creates a circle of food with reducing amounts around a coordinate |
| [param] radius – The radius of the food | integer |
| affectedCells – The cells which lie in the food blob | [{x: integer, y: integer}] |
| distanceFromCenter – The distance between the cell and the coord | number |
| amount – The food amount | integer |
| FoodSystem.addFood | N/A |  | Adds random sized food blobs at random positions in the map |

### Food

|  |  |  |  |
| --- | --- | --- | --- |
| Function name | Variables | Type | Description |
| Food | this.foodSystem | FoodSystem object | The food system which controls the food pieces |
| this.size | width: number, height: number | The size of the food in pixels (default: CELL\_SIZE) |
| this.amount | integer | The concentration of food the piece has |
| this.coord | x: number, y: number | The coordinate of the piece of food |
| Food.addToMap | N/A |  | Adds the piece of food to the map |
| Food.removeFromMap | N/A |  | Removes the piece of food from the map |
| Food.draw | [param]ctx | Canvas context object | Draws the piece of food onto the canvas context |

## Modules

### Map

| **Function name** | **Variables** | **Type** | **Description** |
| --- | --- | --- | --- |
| drawGrid | [param]ctx | Canvas context object | Draws a grid onto the canvas context |
| drawBackground | [param]ctx | Canvas context object | Draws a rectangle over the entire canvas effectively wiping it to a single colour |
| zoom | [param] level – The zoom level (positive : zoom in, negative : zoom out) | number | Change the simulation zoom level |
| createMap | N/A |  | Create an empty map and populate it with default values |

### Canvas

| Function name | Variables | Type | Description |
| --- | --- | --- | --- |
| resizeElement | [param] element – The element to be resized | HTML element | Resizes a HTML element to a size |
| [param] size – The new size of the element | width: number, height: number |
| drawRect | [param]ctx – The canvas context | Canvas context object | Draws a rectangle onto a canvas context |
| [param]coord – The coordinate of the top left corner | x: number, y: number |
| [param] size – The size of the rectangle | width: number, height: number |
| [param]fillColour – The colour of the rectangle | string |
| [param]strokeColour – The colour of the rectangles boarder (default: ‘#000000’) | string |
| [param]lineWidth – The width of the border (default: 0) | number |
| drawLine | [param]ctx – The context which the line will be drawn onto | Canvas context object | Draws a line onto a canvas context |
| [param] coord1 – The coordinate of the starting point | x: number, y: number |
| [param] coord2 – The coordinate of the end point | x: number, y: number |
| [param]strokeColour – The stroke colour of the line | string |
| [param]lineWidth – The width of the stroke | number |
| drawArc | [param] ctx – The canvas context which the arc will be drawn onto | Canvas context object | Draws an arc onto a canvas context |
| [param]coord – The coordinate of the centre of the arc | x: number, y: number |
| [param] radius – The radius of the arc | number |
| [param]startAngle – The angle from the horizontal to start the arc at (clockwise) | number |
| [param]endAngle – The angle from the horizontal to stop the arc at (clockwise) | number |
| [param]strokeColour – The stroke colour of the arc | string |
| [param]lineWidth – The width of the stroke | number |
| [param]fillColour – The colour of the arc | string |
| drawCircle | [param] ctx – The canvas context which the circle will be drawn onto | Canvas context object | Draws a circle onto a canvas context |
| [param]coord – The coordinate of the centre of the circle | x: number, y: number |  |
| [param] radius – The radius of the circle | number |  |
| [param]fillColour – The colour of the circle | string |  |
| clearCanvas | [param] ctx – The canvas context | Canvas context object | Draws a rectangle over the entire canvas effectively wiping it to a single colour (OUT\_OF\_BOUNDS\_COLOUR) |

### Utilities

| Function name | Variables | Type | Description |
| --- | --- | --- | --- |
| randInt | [param]range – The range inclusive | min: integer, max: integer | Returns a random integer within a specific range  Return integer – The random number |
| randFloat | [param]range – The range inclusive | min: number, max: number | Returns a random float within a specific range  Return number – The random float |
| randDir | N/A |  | Returns a random angle between 0 and 2π radians  Return number – The random direction |
| randColour | N/A |  | Returns a random hexadecimal colour  Return string – The random colour |
| validateDirection | [param]dir – The direction | number | Returns an angle in the range 0 to 2π e.g. if dir = 4π returns 2π  Return number – The new angle |
| randProperty | [param]obj – The object | object | Picks a random property of an object  Return object – The random property from the object |
| distance | [param]coord1, coord2 – The coordinates to find the distance between | x: number, y: number | Returns the shortest distance between two coordinates  Return number – The distance |
| x – The distance in the x axis | number |
| y – The distance in the y axis | number |
| scaleCoord | [param]coord – The coordinate to scale | x: number, y: number | Scale a coordinate to its location in pixels  Return {x: number, y: number} – The scaled coordinate |
| coordToIndex | [param]coord – The coordinate which will be converted | x: number, y: number | Converts a coordinate to a map index, it can be used with coordinates which don’t map exactly to a single index and will work out which cell the coordinate lies in mostly  Return integer – The map index which the coordinate converts to |
| indexToCoord | [param]index – The map index | integer | Converts a map index to a coordinate  Return {x: number, y: number} – The coordinate which the index converts to |
| x – The x coordinate of the index | number |
| y – The y coordinate of the index | number |
| getCellCoord | [param]coord – The coordinate | x: number, y: number | Similar to coordToIndexhowever returns the *neat* coordinate i.e. a coordinate which maps exactly to a single map index by finding the cell which the coordinate lies in mostly  Return {x: number, y: number} – The coordinate which maps exactly to a single map index |
| boundary | [param]coord – The coordinate to test | x: number, y: number | Returns the *wrapped* coordinate of a coordinate if it exceeds the map boundary. As the map wraps around if an ant walks off one side it will appear on the other  Return {x: number, y: number} – The wrapped coordinate |
| [param]bounds – The boundary which is tested against | x: {min: number, max: number}, y: {min: number, max: number} |
| getBlock | [param]coord – The coordinate to get the block around | x: number, y: number | Returns an array of cells which lie a certain distance around a specific point  Return {x: number, y: number} – An array of coordinate which lie around the coordinate |
| [param]size – The size of the block i.e. if width = 2, takes 2 blocks to the left and two blocks on the right of the coordinate | width: integer, height: integer |
| getSector | [param]coord – The coordinate to get the block around | x: number, y: number | Returns an array of cells which lie in the sector of a circle of a particular radius around a specific point  Return [{x: number, y: number}] – An array of coordinates which lie in the sector |
| [param]radius – The radius of the sector | number |
| [param]direction – The direction the ant is facing | number |
| [param]angle – The angle of the sector | number |
| searchCoord – The coordinate of the cell currently being checked to see if it lies in the sector | x: number, y: number |
| turnAround | angle – The angle in radians | number | Returns the reverse direction  Return number – The reversed direction |
| angleTo | [param]coord, target – The coordinates to find the angle between | x: number, y: number | Returns the direction/angle of shortest path to get from the coord to the target  Return number – the angle from the vertical axis clockwise in radians |
| createAnt | [param]species – The species of the new ant | Species object | Creates a new ant |
| [param]coord – The coordinate of the new ant | x: number, y: number |
| [param]nest – The new ants home nest | Nest object |
| [param]startingHealth – The new ants health | number |
| [param]type – The type of ant to create e.g. ANT\_TYPE.worker | integer |
| genID | N/A |  | Generates a unique id. Requires CURRENT\_ID variable to keep track of current id  Return integer – A unique ID |
| clone | [param]obj – The object which will be cloned | object | Clones an object *Note*: needed as JavaScript passes everything by reference  Return object – A copy of object |
| temp – A copy of the object | object |
| getElement | [param]id – The HTML id | string | Returns the HTML element on ID responds to.  Return HTML element object – The HTML element |
| setValue | [param]id – The HTML id | string | Sets the value of a HTML element |
| [param]value – The value which the HTML element is being set to | string |

### Controls

| Function name | Variables | Type | Description |
| --- | --- | --- | --- |
| start | N/A |  | Starts the simulation |
| pause | N/A |  | Stops the simulation |
| toggleRunning | N/A |  | Toggles between running and paused |
| step | N/A |  | Steps through a single tick of the simulation |
| runPauseButton | N/A |  | Toggles running and updates the pause/run button |
| updateValue | [param] element – An element of an input in the settings panel | HTML element object | Updates the value of a characteristic i.e. when a user slides a slider, the characteristics value is updated |
| [param] value – The new value of the characteristic | string |
| characteristic – The characteristic being updated | object |
| speciesCost – The cost of the species with the updated value | number |
| workerFoodCost, queenFoodCost, soldierFoodCost – The costs of each type of ant | number |
| queenStepsMax, queenStepsMin – The number of queen steps (used to determine if the min is > then the max) | number |
| updateDefaultValues | N/A |  | Updates the value of all characteristics of the selected species to their default values |
| characteristic – The characteristic being changed | object |
| updateRandomValues | N/A |  | Updates the value of all characteristics of the selected species to random values |
| characteristic – The characteristic being changed. | object |
| value - the new value of the characteristic | number |
| newElement | [param]tag – A HTML tag name e.g. “div” | string | Creates a new HTML element  Return HTML element – The new element |
| [param]attribute – An array of attributes to add to the new element | [{type: string, value: \*}] |
| element – The new HTML element |  |
| createInputType | [param] characteristic – A single characteristic from CHARS | object | Creates an input element e.g. a range input or a button input |
| [param] prop – The property of CHARS which the characteristic refers to | string |
| input – the input element | HTML element |
| createInput | [param] characteristic – A single characteristic from CHARS | object | Creates a row containing a label, input and value elements used for creating dynamic inputs for characteristics  Return HTML element – The row containing the label, input and value |
| [param] prop – The property of CHARS which the characteristic refers to | string |
| row – The row in the table which will contain the characteristic | HTML element |
| label – The table item containing the characteristics label | HTML element |
| inputContainer – The container which will hold the input. Needed to scale the input correctly with CSS | HTML element |
| input – The input element | HTML element |
| value – The table item containing the current value of the input | HTML element |
| createCharacterisitcInputs | N/A |  | Creates inputs for all characteristics used for creating dynamic inputs for characteristics. These are all appended to a table in the configuration panel |
| configPanel – The div which holds all elements in the configuration panel | HTML element |
| table – The table holding all the characteristic | HTML element |
| inputRow – The row returned by createInput | HTML element |
| updateUserSpecies | N/A |  | Updates the selected species with the new values selected in the configuration panel i.e. what happens when the update button is pressed |
| speciesCost – The total cost of the species in health | number |
| characteristic – The current characteristic being updated | object |
| createDataRow | [param]className – The name of the class for the species | string | Creates a row containing a label and data elements used for displaying information about species  Return HTML element – The data row |
| [param] id – The ID which will be used to access and update the species | string |
| [param]labelValue – The text which will be used as a label for the data | string |
| [param]dataValue – The data which will be displayed | string |
| row – The HTML row element which will contain the label and data | HTML element |
| label – The table item which contains the labelValue text | HTML element |
| data – The table item which contains the dataValue text | HTML element |
| createSpeciesData | [param] species – The species whose data you want to create a data display for | Species object | Create all elements to display information about a species |
| id – The species ID | string |
| className – The name of the class which responds to all items within a species data | string |
| table – The table which will contain all of the data | HTML element |
| titleRow - the HTML row which contains the title | HTML element |
| title – The title showing which species the table refers to | HTML element |
| toggleVisibility – The button which is used to show and hide the species data | HTML element |
| colourRow – The row containing information about the species colour | HTML element |
| antNumRow – The row containing information about the number of ants in the species | HTML element |
| nestNumRow – The row containing information about the number of nests in the species | HTML element |
| foodAmount – The amount of food all the nests in the species have combined | number |
| foodAmountRow – The row containing information about the amount of food the species has i.e. all of the nests combined | HTML element |
| button – Used to toggle the visibility of the data | HTML element |
| updateSpeciesData | N/A |  | Updates data for all of the species currently in the simulation i.e. so that the information in the data panel reflects the current state of the species in the simulation |
| species – The species being updated | Species object |
| id – The ID of that species | string |
| colourDataElement – The element which contains information about the species colour | HTML element |
| antNumDataElement - The element which contains information about the species number of ants | HTML element |
| nestNumDataElement - The element which contains information about the species number of nests | HTML element |
| foodAmountDataElement - The element which contains information about the species amount of food in all nests | HTML element |
| foodAmount – The amount of food contained in each nest in the species combined | number |
| removeSpeciesData | [param] id – The id of the species to remove | string | Removes a species data i.e. when it has died out |
| nextVisibility | [param] button – A reference to the toggle visibility button | HTML element | Checks whether the species data id currently maximised or minimised and returns whether or not the species data should be hidden or not assuming the button has been clicked  Return string – The value of the displaying styling |
| toggleVisibilityButton | [param] button – A reference to the toggle visibility button | HTML element | Toggles the text in the visibility button |
| toggleClassVisibility | [param] button – A reference to the toggle visibility button | HTML element | Maximises and Minimises the data for a particular species |
| id – The id of the species the toggle visibility button corresponds to | string |
| listOfClassElements – All elements of the same class i.e. all elements in the species data to be hidden | [HTML element] |
| select | [param] title – A reference to the title element of the species data | HTML element | Selects a new species allowing the user to alter that specie’s characteristics |
| id – The id of the species | string |
| species – The species being selected | Species object |
| characteristic – The characteristic being updated | HTML element |

### Main

| Function name | Variables | Type | Description |
| --- | --- | --- | --- |
| tick | N/A |  | Performs actions required each tick |
| drawMap | [param]ctx – The canvas context | Canvas context object | Draws all objects onto the canvas and also updates all objects |
| ant – The ant to draw | Ant object |
| pheromone – The pheromone to draw | Pheromone object |
| food – The piece of food to draw | Food object |
| createEnviroment | N/A |  | Setups the environment when starting or restarting the simulation |
| simulationFoodSystem – The food system which controls the placement of food in the map | FoodSystem object |
| x – The random x-coordinate of the starting queen | integer |
| y – The random y-coordinate of the starting queen | integer |  |
| window.onload (Anonymous function) | N/A |  | Performs actions once the page is loaded i.e. all scripts are loaded and the canvas is loaded |
| start – The coordinate of the start of the drag | x: number, y: number |
| end – The coordinate of the end of the drag | x: number, y: number |
| window.onkeydown (Anonymous function) | [param] e – The event | Event object | A call back function for when a key down press event happens. Controls what each key does e.g. arrow keys pan |
| charCode – The character code the event maps to | string |
| canvasDOM.addEventListener(‘mousewheel’) (Anonymous function) | [param] e – The event | Event object | Controls scrolling in the map when the mouse wheel is used (Chrome and IE only) |
| delta – The amount scrolled | number |
| canvasDOM.addEventListener(‘DOMMouseScroll’) (Anonymous function) | [param] e – The event | Event object | Controls scrolling in the map when the mouse wheel is used (Firefox only) |
| delta – The amount scrolled | number |
| canvasDOM.onmousedown (Anonymous function) | [param] e – The event | Event object | Controls panning behaviour for when the mouse is down |
| canvasDOM.touchstart (Anonymous function) | [param] e – The event | Event object | Controls panning behaviour when the canvas is touched |
| canvasDOM.onmousemove (Anonymous function) | [param] e – The event | Event object | Controls panning behaviour when the mouse is being moved |
| canvasDOM.touchMove (Anonymous function) | [param] e – The event | Event object | Controls panning behaviour when a finger is moved across the simulation |
| canvasDOM.mouseup (Anonymous function) | [param] e – The event | Event object | Stops panning the simulation |
| canvasDOM.touchend (Anonymous function) | [param] e – The event | Event object | Stops panning the simulation |

| Global variable | Type | Description |
| --- | --- | --- |
| canvasDOM | HTML element | The canvas HTML element |
| canvasCTX | Canvas context object | The canvas elements context (used for drawing to) |
| lastLoop | number | The time taken for the last tick to complete in milliseconds |
| BUTTONS | object | Holds the actions that buttons do e.g. when the reset button is pressed (must have id = ‘reset’) call the updateDefaultValues function would be BUTTONS = {reset: updateDefaultValues} |
| draging | boolean | Holds the current state of the simulations pan i.e. if the simulation is currently being dragged by a mouse or finger. (default: false) |

### Config

| Global variable | Type | Description |
| --- | --- | --- |
| VALUE\_TYPE | object (constant) | Used to signify if a variable is an integer or a float i.e. needed as some characteristics require integer values while others require floats and so to generate random values for these characteristics must know what type they need |
| TICK | number (constant) | The time between ticks (default: 10) |
| CURRENT\_ID | number | Used to keep track of the latest unique ID (default: 0) |
| ANTS\_LIST | [Ant object] | Holds all ant objects, used to update all ants (default: []) |
| SPECIES\_LIST | [Species object] | Holds all species objects, use to update all species (default: []) |
| RUNNING | boolean | Determines whether the simulation is running or not (default: false) |
| GRID\_COLOUR | string (constant) | The colour of the maps grid lines (default: ‘#000000’) |
| GRID\_LINE\_WIDTH | number (constant) | The width of the grid lines (default: 0.2) |
| BACKGROUND\_COLOUR | string (constant) | The canvas background colour (default: ‘#C2ABBA’) |
| OUT\_OF\_BOUNDS\_COLOUR | string (constant) | The colour of any area off map in the simulation (default: ‘#FFFFFF’) |
| CELL\_SIZE | width: integer, height: integer | Size in pixels of a single cell on the map (default: width: 6, height: 6) |
| CANVAS | name: string, width: integer,  height: integer | Defines characteristics about the simulations canvas (default: name: ‘simulation’, width: 500, height: 500) |
| AVERAGE\_FOOD\_SAMPLE\_RATE | integer (constant) | The number of ticks to wait between sampling for averages of food (default: 10) |
| NUMBER\_OF\_FIXED\_PLACES | integer (constant) | The number of numbers after the decimal place to display when displaying values on the interface (default: 2) |
| SELECTED\_SPECIES | Species object | Holds the species whose characteristics can be changed in the characteristics panel |
| SELECTED\_COLOUR | string (constant) | The colour of the selected species in the data panel (default: ‘#A5C7D9’) |
| UNSELECTED\_COLOUR | string (constant) | The default colour for unselected species in the data panel (default: ‘#FFFFFF’) |
| BUTTON\_UPDATE\_COLOUR | string (constant) | The colour of the update button when the characteristics of the selected species have been changed by the user (‘#FF0000’) |
| BUTTON\_NO\_UPDATE\_COLOUR | string (constant) | The normal colour of a button when no settings have been altered (default: ‘#000000’) |
| LEFT\_ARROW\_KEY | integer (constant) | The char code of the left arrow key (default: 37) |
| RIGHT\_ARROW\_KEY | integer (constant) | The char code of the right arrow key (default: 39) |
| UP\_ARROW\_KEY | integer (constant) | The char code of the up arrow key (default: 38) |
| DOWN\_ARROW\_KEY | integer (constant) | The char code of the down arrow key (default: 40) |
| PLUS\_KEY | integer (constant) | The char code of the plus key (default: 107) |
| MINUS\_KEY | integer (constant) | The char code of the minus/dash key (default: 109) |
| SPACE\_BAR\_KEY | integer (constant) | The char code of the space bar key (default: 32) |
| R\_KEY | integer (constant) | The char code of the ‘r’ key (default: 82) |
| S\_KEY | integer (constant) | The char code of the ‘s’ key (default: 83) |
| MIN\_ZOOM | number (constant) | The minimum zoom amount (default: 1) |
| MAX\_ZOOM | number (constant) | The maximum zoom amount (default: 14) |
| PAN\_AMOUNT | integer (constant) | The number of pixels to pan by for each key press of an arrow key (default: 15) |
| ZOOM\_AMOUNT | number (constant) | The amount to zoom for each key press of the plus or minus key (direction depends on key press i.e. plus is zoom in while minus is zoom out) (default: 0.5) |
| CANVAS\_OFFSET | x: integer, y: integer | The amount of pixels the simulation on the canvas is offset by from the top left corner of the simulation (default: x: 0, y: 0) |
| INPUT\_TYPE | object (constant) | Used for creating custom inputs i.e. sliders or buttons |
| MAP | [objects] | Holds all objects displayed on the map (default: []) |
| GRID\_SIZE | width: integer, height: integer | Size in number of cells of the map (default: width: 250, height: 250) |
| MAP\_BOUNDARY | x: {min: integer, max: integer}, y: {min: integer, max: integer} | Used to define the edge of the map i.e. when to wrap the coordinate of the ant e.g. if ant exceeds x.max place the ant on the other side of the map as if the map was a torus |
| NUM\_OF\_CELLS | integer | The total number of cells in the map |
| FOOD\_COLOUR | string (constant) | The colour of food in the simulation (default: ‘#00FF00’) |
| FOOD\_HEALTH\_RATIO | number (constant) | food : health i.e. if = 50 then 1 piece of food is worth 50 health (default: 50) |
| FOOD\_CHANCE | number (constant) | The probability of a food source in a cell (default: 0.0004) |
| STARTING\_QUEEN\_ANT\_NUMBER | integer | The number of queen ants at the start of the simulation (default: 1) |
| ANT\_FOOD\_DROP\_SPEED | integer (constant) | The number of ticks it takes for an ant to drop a piece of food (default: 15) |
| ANT\_FOOD\_TAKE\_SPEED | integer (constant) | The number of ticks it takes for an ant to pick up a single piece of food (default: 30) |
| DAMAGE\_MULTIPLIER | number (constant) | The amount of health a soldier ant takes does to ants its attacking i.e. if = 100 then 100 \* jawStrength of soldier is the amount of health the ant being attacked would lose each tick it is attacked (default: 100) |
| NEST\_GUARD\_RADIUS | integer (constant) | The number of steps ants take once seeing the nest i.e. the sentry radius of solider ants about the nest (default: 100) |
| SOLDIER\_ANT\_MAX\_TARGET\_DISTANCE | integer (constant) | The maximum distance between an ant and its target before losing interest (default: 25) |
| TURN\_RATE | number (constant) | The rate at which ants turn when guarding i.e. 0.02 means the ant turns 0.02 radians each tick (default: 0.02) |
| GOAL | object (constant) | Ant’s goals used to determine what actions the ants need to perform |
| NEST\_SIZE | width: integer, height: integer (constant) | The extra size of a nest in cells e.g. {width: 1, height: 1} means go 1 extra cell in both sides and 1 extra cell both up and down (default: width: 1, height: 1) |
| ANT\_TYPE | object (constant) | Used to show the type of ant |
| PHEROMONE\_EVAPERATION\_RATE | number (constant) | The rate at which pheromones in the simulation lose concentration per tick (default: 0.005) |
| MAX\_PHEROMONE\_CONVENTRATION | number (constant) | The maximum concentration a pheromone can have (default: 1) |
| USER\_SPECIES | Species object | The species used on the first run of the simulation i.e. the first species |
| CHARS | Object | Holds properties of all characteristics species:   * min – number – The minimum value a characteristic can be * max – number – The maximum value a characteristic can be * type – integer – The type of value a characteristic is i.e. float or integer (use VALUE\_TYPE to define e.g. VALUE\_TYPE.floatValue) * id – string – The HTML ID property of the characteristic * neatName – string – The name used in a label in the HTML * desc – string – The description of what the characteristic does * step – number – The minimum change in the characteristics value * healthModifier – number – The cost of the characteristic (= healthModifier \* value) * defaultValue – number – The default value of the characteristic * value – number – The actual value of the HTML input (differs from the value stored in species as this is formatted for displaying i.e. to a certain number of decimal places) * editable – boolean – Determines if a characteristic can be edited (true if it can be edited else false) * inputType – integer – The type of input required e.g. button (Use INPUT\_TYPE to define e.g. INPUT\_TYPE.slider) |

### Difficult to understand code

This section contains code snippets which may not be clear at first look in the code.

*Note*: comments omitted to reduce size.

### The addToMapfunction

The addToMap function is used in multiple classes for example in Ant:

Ant.prototype.addToMap **=** **function**() {

**if** (**this**.alive) MAP[getCellIndex(**this**.coord)].ant.push(**this**);

};

It adds an ant object to the MAP at the index which the ants coordinate maps to if the ant is still alive. Objects can be added to either MAP[index].ant, MAP[index].food or MAP[index].pheromonedepending on the type of object being added.

### The Ant.wonder function

The Ant.wonder function is used by ants when the ant is moving around the map with no specific target e.g. if the ant is searching for something such as when workers are looking for food.

*Note*: A flowchart of this algorithm can be found in the algorithms section in design. No major differences exist between the flowchart and the implemented code.

Ant.prototype.wonder **=** **function**() {

**var** M **=** 0;

**var** Mxy **=** {

        x**:** 0,

        y**:** 0

    };

**var** CoM **=** {

        x**:** 0,

        y**:** 0

    };

**var** pheromones **=** false;

**for** (**var** i **=** 0; i **<** **this**.pheromonesInRange.length; i**++**) {

**if** (**this**.pheromonesInRange[i].species **===** **this**.species) {

            M **+=** **this**.pheromonesInRange[i].concentration;

            Mxy.y **+=** **this**.pheromonesInRange[i].coord.y **\*** **this**.pheromonesInRange[i].concentration;

            Mxy.x **+=** **this**.pheromonesInRange[i].coord.x **\*** **this**.pheromonesInRange[i].concentration;

            pheromones **=** true;

        }

    }

    CoM **=** {x**:** Mxy.x **/** M, y**:** Mxy.y **/** M};

**if** (Math.random() **<** **this**.species.chars.explorativeInfluence)

**this**.prioritizeDirection **=** randDir();

**if** (pheromones **&&** Math.random() **<** **this**.species.chars.pheromoneInfluence) {

**var** angle **=** angleTo(**this**.coord, CoM);

angle **=** validateDirection(angle);

**if** (angle **>** **this**.direction **-** Math.PI **/** 3 **&&** angle **<** **this**.direction **+** Math.PI **/** 3) {

**this**.direction **=** angle;

**this**.prioritizeDirection **=** **this**.direction;

**this**.followingPheromone **=** true;

        } **else** {

**this**.direction **=** **this**.prioritizeDirection;

**this**.followingPheromone **=** false;

        }

    } **else** {

**this**.direction **=** **this**.prioritizeDirection;

**this**.followingPheromone **=** false;

    }

};

The functions purpose is to determine which direction an ant should move, ants are more likely to move towards pheromones. First, the mean direction of all the pheromones in range is calculated weighted by the pheromones concentration CoM (e.g. closer pheromones with a large concentration will have a large effect on the direction then further away pheromones with a lower concentration).

After CoM is calculated there is a chance that the ant will change their prioritizedDirection to a random value. The prioritizeDirection is the direction the ant wants to move in, it is needed so that when an ant is not being influenced by a direction that the ant will move in straight lines rather than just completely randomly.

The ant will have a chance of following the mean pheromone direction, however only if the mean direction is not more than radians from the current direction, this stops ants turning 120° or more around and following a pheromone from the direction they just came. Finally if none of these conditions are true, the ant will follow the prioritizeDirection.

### The die function

The die function is used in a very similar way in multiple classes throughout the simulation; here is how it is implemented in the Ant class:

Ant.prototype.die = **function**() {

**var** index = ANTS\_LIST.indexOf(**this**);

ANTS\_LIST.splice(index, 1);

**var** index = **this**.species.ants.indexOf(**this**);

**this**.species.ants.splice(index, 1);

**this**.alive = false;

};

The point of the function is to stop the simulation displaying an object and also remove all references to an object so that garbage collections will destroy the object. To stop an object from being displayed it must not be added to the map, this is the purpose of **this**.alive = false; and **if** (**!this**.alive) **return** void(0); which will stop the object being re added to the map after removal.

To stop the object updating its references must be removed, all ants are stored in ANTS\_LIST (*Note*: both food and pheromones are stored directly on the map), ants must also be removed from **this**.species.ants so that the species has an accurate number of live ants.

### Food vs. Health

Food on the map contains different amounts of food, when a worker ant collects food it takes a certain amount of food from a food piece and adds it to **this**.carrying. A single amount of food is worth a certain amount of health, this amount is determined by FOOD\_HEALTH\_RATIO. So if a worker eats one piece of food its health will increase by FOOD\_HEALTH\_RATIO. So food is collected by worker ants from the map and converted into health.

### Nest reproducing health

Species have a specific health cost depending on how favourable a specie's characteristics are. Ants also required a specific amount of health to be created (this is a characteristic of their species). When a nest creates an ant, the ant will be born with an amount of starting health, this value is the amount of health needed to create the ant minus the cost the species.

### The reproduce function

The reproduce function is implemented in the species class. It determines which type of ant to create.

*Note*: A flowchart of this algorithm can be found in the algorithms section in design. No major differences exist between the flowchart and the implemented code.

Nest.prototype.reproduce **=** **function**() {

**var** prob **=** Math.random();

**var** chars **=** **this**.species.chars;

*// Normalize probabilities and then sort into ascending order*

**var** sum **=** chars.reproductionQueenProb **+** chars.reproductionSoldierProb **+**

chars.reproductionWorkerProb;

**var** queenProb **=** chars.reproductionQueenProb **/** sum;

**var** soldierProb **=** chars.reproductionSoldierProb **/** sum;

**var** workerProb **=** chars.reproductionWorkerProb **/** sum;

**var** ordered **=** [{

prob**:** queenProb,

type**:** ANT\_TYPE.queen

}, {

prob**:** soldierProb,

type**:** ANT\_TYPE.soldier

}, {

prob**:** workerProb,

type**:** ANT\_TYPE.worker

}].sort(**function**(a, b) {

**return** a.prob **-** b.prob;

}); *// sort min to max*

*// Determine which outcome occurred*

**if** (prob **<** ordered[0].prob **&&** **this**.viable(ordered[0].type))

**this**.createAnt(ordered[0].type);

**else** **if** ((prob **<** ordered[1].prob **+** ordered[0].prob) **&&** *// cumulative probability*

**this**.viable(ordered[1].type))

**this**.createAnt(ordered[1].type);

**else** **if** (prob **<** ordered[2].prob **+** ordered[1].prob **+** ordered[0].prob **&&**

**this**.viable(ordered[2].type))

**this**.createAnt(ordered[2].type);

};

This function is called to decide which type of ant to create, at first the ants probability of each type of ant (worker, queen and soldier) are normalized to a sum of one. The normalized values are then ordered into ascending order. A random number is then compared with the normalized values, to determine which probability the random number lies in (*Note*: the probabilities are cumulative as they are normalized thus are added). At each comparison the viability of the ant type is determined, this determines if the nest can afford to create that type of ant i.e. the nest may not have enough health to create a queen and therefore cannot create one. Once the probability which the random number falls into is found and the ant type is viable, the ant which the probability maps to is created.

### Difference between Nest and NestPiece classes

The Nest is an object which represents an ant nest; it contains the amount of food the workers deposit into it. A NestPiece is an object used to represent part of a nest on the MAP. Individual nest pieces can be destroyed by soldier ants however the Nest object will continue to exist until there are no NestPieces left.

### Goals

Ants’ actions are decided by the goal they have been assigned, goals are results which want to be achieved, they often require multiple tasks to be completed. For example a worker may have the goal getFood, this requires the ant to move towards the food, and over a series of ticks pick up individual pieces of food until it cannot carry any more.

All Goals are located in the GOAL constant set in the configuration file. Not all ants can be assigned all goals. Ants are required to have logic in the doTask and updateGoal functions in order to achieve goals.

### The steps variable

The steps variable is used both in queen ants as well as soldier ants. Steps is used to move an ant a certain amount of steps in the direction it is facing. Its value is often small so that an ant will stay in close to proximity to its original location; this is how it is used in the soldier class. When guarding a nest, if a soldier sees the nest, it will move NEST\_GUARD\_RADIUS steps away, which allows it to stay in proximity to the nest without being in direct line of sight of the nest.

Steps is used in the queen class to find a location for a new nest. A queen ant will move a certain number of steps in a specific direction and once it has moved the number of steps required it will create a nest.

### The guardNest function

This function is only used by soldier ants and contains the logic soldier ants use to guard the nest.

*Note*: A flowchart of this algorithm can be found in the algorithms section in design. No major differences exist between the flowchart and the implemented code.

Soldier.prototype.guardNest **=** **function**() {

**if** (**this**.nearNest) { *// If the ant is close to the nest*

**if** (**this**.steps **<=** 0 **&&** **!this**.soldiersInView()) { *// and no soldiers in view*

**this**.moving **=** false;

**this**.direction **+=** 0.02; *// slowly turn i.e. observing surroundings*

} **else** **if** (**this**.soldiersInView()) { *// If soldiers in view, keep moving*

**this**.nearNest **=** false;

}

} **else** **if** (**this**.seeNest() **&&** **!this**.atNest()) { *// If near the nest, move a*

*// specific number of steps away*

**this**.nearNest **=** true;

**this**.steps **=** NEST\_GUARD\_RADIUS;

} **else** { *// Otherwise, keep looking for the nest*

**this**.wonder();

**this**.moving **=** true;

}

};

When an ant is given the goal to guard its nest (GOAL.guardNest) the ant will try to first find its nest. The ant will wonder around until it is near the nest. The if statement inside the first if block decides whether to stop moving or not. If the ant is not walking away from the nest (i.e. this.steps <= 0) and there are no other friendly soldier ants in view the ant should stop, as its current position is considered a good sentry position. However if there are soldiers in view, the ant should find a new sentry position. This is because to maximise coverage of nests perimeter it makes sense for ants to not double up on positions i.e. at no point having two sentries together as they can both see the same area, it makes more sense to spread out sentries as much as possible i.e. so they cannot see the other sentries.

The **else if** (**this**.seeNest() **&&!this**.atNest()) is used to set a guard radius. If the conditions apply i.e. the ant can see the nest however is not standing on top of the nest, it should move a certain distance away from the nest to find a good sentry position.

Finally if this is not true the ant should keep looking for the nets.

### The mutate function

Is implemented in the species class and used to mutate a species to represent random mutation of genes in mutation when genetic information is passed on.

Species.prototype.mutate = **function**() {

**if** (Math.random() <= **this**.mutationRate) {

**var** altChars = clone(**this**.chars);

**var** altCharacteristic = randProperty(altChars);

**var** altValue = **this**.mutateChar(altCharacteristic);

altChars[altCharacteristic] = altValue;

**var** species = **this**.createSpecies(altChars);

**return** species;

} **else** {

**return** **this**;

}

};

The function picks a single random characteristic to mutate, and will mutate that characteristic. The function will then create and return a new species which is the same as the current one however with a single mutation in one characteristic.

### this.sleep variable

The this.sleep variable is used by the Ant class. It is used to stop the ant from performing an action for a number of ticks. The value of this.sleep is the number of ticks to wait. An example of its use is in the Ant.takeFoodand Ant.movefunctions

Ant.prototype.takeFood **=** **function**(food) {

**if** (**this**.isFood(food)) { *// If food*

        food.amount **-=** 1; *// Take a single piece of food*

**this**.sleep **+=** ANT\_FOOD\_TAKE\_SPEED;

**if** (**!this**.isFood(food)) *// If food is all gone remove it from the map*

food.removeFromMap();

**return** 1;

    } **else** {

**return** 0;

    }

};

Ant.prototype.move **=** **function**() {

**if** (**this**.sleep **<=** 0) { *// only move when not waiting for a task to complete*

**this**.coord.x **+=** Math.sin(**this**.direction) **\*** **this**.species.chars.speed;

**this**.coord.y **-=** Math.cos(**this**.direction) **\*** **this**.species.chars.speed;

    }

    boundary(**this**.coord, MAP\_BOUNDARY); *// Make sure if the ant is out of bounds*

};

Here it is used to stop the ant from moving for a certain amount of time for each piece of food it collects. This leads to more realistic looking behaviour as it shows that the ant takes time to collect food. If there was no sleep timer the ant would be able to walk over food and be able to pick it up instantly. This would reduce the realism of the simulation.

## Configuration

### Simulation

All aspects of the simulation can be easily configured by editing config.js.

### Environment

The simulation environment includes how much food is spawned into the environment and how pheromones change over time. The amount of food is controlled using FOOD\_CHANCE this is the probability of a food source being spawned in a cell, increase this for more food. The rate at which food will grow is controlled by FOOD\_GROW\_RATE i.e. this is the probability each tick that food in the map will grow. The amount that it grows by is controlled by FOOD\_GROW\_AMOUNT again this is a probability which each piece of food is tested against to determine if the piece of food will grow or not. What's more, if the maximum or minimum size of food needs to be changed it can be easily changed within core\main.js by editing the FoodSystem objects properties variation.min and variation.max. Pheromones within the system evaporate over time, the amount each pheromone evaporates is controlled by PHEROMONE\_EVAPERATION\_RATE. There is also a maximum limit on pheromones i.e. the maximum concentration a pheromone can be, it can be changed by changing MAX\_PHEROMONE\_CONCENTRATION. The simulation starts with a single queen ant of the default species however this can be changed by altering STARTING\_QUEEN\_ANT\_NUMBER, this is useful as it can speed up the development of species during the start of the simulation.

### Multipliers

There are two currencies within the simulation, health and food. The food can only be picked up by worker ants and is transported to the nest where it is converted into health. Health is a measure of how long an ant will live for, it slowly decrease over time until it reaches 0 and the ant dies. The rate of decrease is controlled independently for different ants (as some ants may be more efficient than others at conserving health) to edit the rate of decrease of health in a nest for example, edit the Nest object changing its healthRate property. The amount of health a single piece of food is worth is controlled by the FOOD\_HEALTH\_RATIO configuration variable. Another similar multiplier is the DAMAGE\_MULTIPLIER which sets how much health a single value in jawSize takes away from an ant being attacked i.e. the amount of health taken = DAMAGE\_MULTIPLIER \* jawSize.

### Global characteristics

These are characteristics of ants which are the same for all ants within the simulation.

* ANT\_FOOD\_TAKE\_SPEED - This is the number of ticks it takes for an ant to pick up a single piece of food. Useful to change if you want to be able to see ants spending a longer time collecting food.
* ANT\_FOOD\_DROP\_SPEED - Similar to ANT\_FOOD\_TAKE\_SPEED however this is the number of ticks it takes for an ant to drop a single piece of food off at the nest. This models how in ant nests there is a specific place where food is collected and it may take some time for an ant to navigate to it.
* NEST\_GUARD\_RADIUS - This is the number of steps a solider ant will take away from its nest when guarding it.
* SOLDIER\_ANT\_MAX\_TARGET\_DISTANCE - This is the maximum distance a target ant can be away from a solider ant chasing it before the solider ant gives up interest of chasing it, and picks a new target.
* NEST\_SIZE.width and NEST\_SIZE.height - The number of extra cells around the nest which the nest occupies.

### Species characteristics

The following is a list of the properties each characteristic has:

* min - The minimum values a characteristic can be.
* max - The maximum values a characteristic can be.
* type - The type of value a characteristic is i.e. a float or integer
* id - The HTML ID property of the characteristic. (Useful to edit if want custom styling for certain characteristics).
* neatName - The name used in a label in the HTML.
* desc - Describe what the characteristic does (used in hover text).
* step - The minimum change in the characteristics value.
* healthModifier - The cost of the characteristic (= healthModifier \* value).
* defaultValue - The default value.
* value - The actual value of the HTML input (differs from the value stored in species as this is formatted for displaying i.e. may have reduced decimal places).
* editable - Determines if a characteristic can be edited by the user.
* inputType - The type of input required i.e. button.

### Interface

To configure the layout of the webpage, edit the index.html and style.css files. This will allow you to change the colour of the webpage or add custom styles. *Note*: core\controls.js can also be edited to change the layout of the configuration and data panels.

### Colours

The following colours can be edited:

* BACKGROUND\_COLOUR - Simulations background colour.
* GRID\_COLOUR - The grid colour.
* OUT\_OF\_BOUNDS\_COLOUR - The out of bounds colour.
* SELECTED\_COLOUR and UNSELECTED\_COLOUR – The selected species colour.
* BUTTON\_UPDATE\_COLOUR and BUTTON\_NO\_UPDATE\_COLOUR - The update button colours.
* FOOD\_COLOUR - The colour of food.

### Size

The size of the simulation can also be edited. Simply edit CANVAS.width and CANVAS.height variables to change the size of the simulation (*Note*: this may require some changes to style.css to make it look correct). Both the grid size and the size of individual cells can be changed by editing GRID\_SIZE.width, GRID\_SIZE.height and CELL\_SIZE.width, CELL\_SIZE.height respectfully.

### Controls

The simulation has a number of controls which can be changes. The tick rate is the time it takes for a single loop of the entire program, it can be changed by editing the TICK variable (the time in milliseconds of a single tick). Furthermore, zooming and panning can be updated with MIN\_ZOOM, MAX\_ZOOM, ZOOM\_AMOUNT and PAN\_AMOUNT. All shortcuts e.g. space bar to pause the simulation can be easily changed in the core\main.js file, by altering the window.onkeydown case statement.

## Test driven development

Parts of the system have been built using test driven development principles. This means there are a large number of automatic tests written for the system. These will automatically determine if functions are working as expected. These can be run after altering the application in any way to determine if the alteration has broken any other parts of the system. This is clearly useful to more easily track to stops bugs appearing. It is recommended to re run all tests after every major change to the application, this will reduce the number of bugs and errors in the program by highlighting what breaks early in the development.

### Run a test

Uncomment the test in index.html i.e. remove<!— and--> which prepend and postponed the script tag. When the test runs it will output the results of the test in the developer console in the browser.

When running tests make sure to add:

<script src="tests/equal.js" type="text/javascript"></script>

<script src="tests/test.js" type="text/javascript"></script>

<script src="tests/testCase.js" type="text/javascript"></script>

To the index.html so that tests can run.

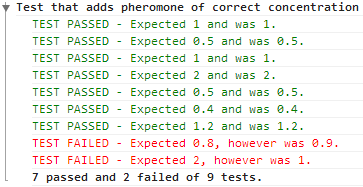
*Note*: An external custom written testing framework is used, to perform tests. As this is not part of the tool and is only used as part of the testing process it is not documented in this document. However its code will be included in the appendix.

### C:\Dropbox\projects\Ant-Simulation\writeup\assests\Maintainance\Testing\Passed distance test.PNGA passed test

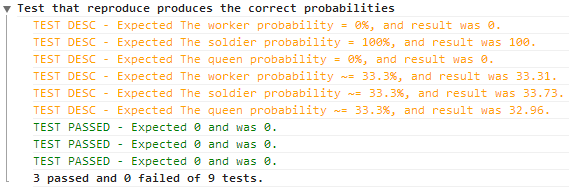
This is an example of the output of a test for the distance function which calculates the distance between two points. For full documentation on a test view the tests source file located in the tests/ directory.

### A failed test

This is an example of the Ant.secrete test failing.



### Human input



Not all tests are completely automatic, some tests require human analysis. Tests which require human analysis are highlighted in. This is testing that the Nest.reproduce function produces the correct proportions of the different types of ants depending on a particular set of characteristics. As this is a probabilistic test (due to the randomness used in the function) there is no deterministic outcome. A probability is used over 10,000 runs to determine the probability for each ant. A human is required to evaluate whether the probability is sufficient to accept the test has passed or not.

# Section 6: User Manual

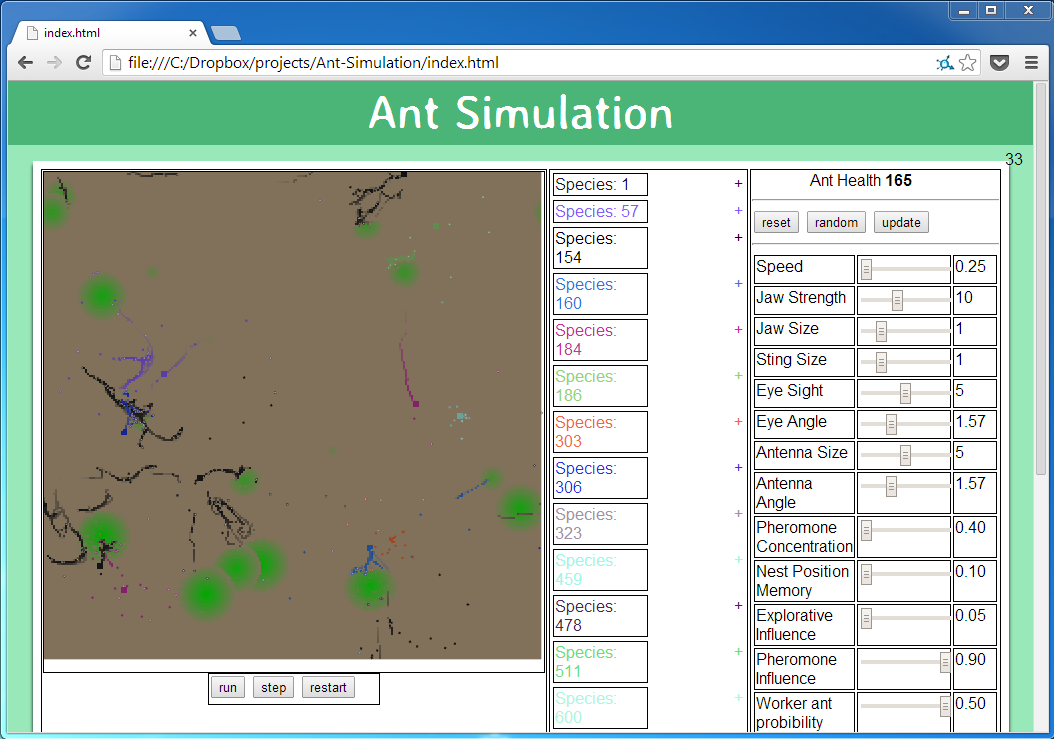
User manual

Ant simulation



## Introduction

The tool is aimed at students studying evolution and natural selection in biology. It is primarily an ant simulation which can be used to see concepts such as survival of the fittest, inheritance, mutation and a number of other ideas around evolution. The tool allows the user to interact with the simulation by the altering of ant species characteristics. This allows the user to design ants and see how they evolve over time, (or die off by getting out competed by better adapted ants). Ants compete for food which is placed randomly on the map. The best adapted ants will be able to collect more food than other ants and so will be able to survive for longer. The tool is a web application and can therefore be accessed within the classroom as well as at home with just a web browser.



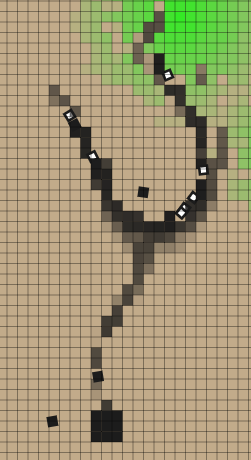
## Simulation overview

This tool simulates how ant’s characteristics affect behaviour and effectiveness of survival. It models three types of ants:

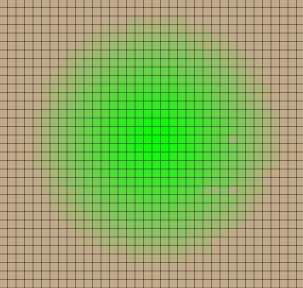
* **Worker ants** – Responsible for finding, collecting and depositing food to the nest.
* **Soldier ants** – Responsible for guarding ants of the same species and also attacking ants and nests from other species.
* **Queen ants** – Responsible for locating a site to create a nest, and then creating a nest.

**C:\Dropbox\projects\Ant-Simulation\assests\User Manual\Ants\Queen ant.PNGC:\Dropbox\projects\Ant-Simulation\assests\User Manual\Ants\Soldier ant.PNGC:\Dropbox\projects\Ant-Simulation\assests\User Manual\Ants\Worker ant (food).PNGC:\Dropbox\projects\Ant-Simulation\assests\User Manual\Ants\Worker ant (no food).PNG***From left to right – A worker ant, a worker ant carrying food, a soldier ant and a queen ant*

The simulation revolves around the ants struggle for food. Food is represented by green blobs. Food is slowly taken away from its original position by worker ants and deposited to the nest. The food is used for creating more ants as well as maintaining the current ants. The darker the colour of the food the more food is concentrated. Food will slowly grow back over time, if it is not all completely gone.



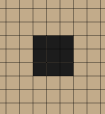
A pheromone trial from food to the nest. It is being followed by a number of ants so returning with food and others looking for food.



A blob of food

Food can be converted into health. Each ant has a certain amount of health which will decrease at a constant rate throughout the simulation. If an ant's health drops below zero the ant will die. To increase an ant’s health the ant can eat food. Food is converted into health with a fixed ratio.

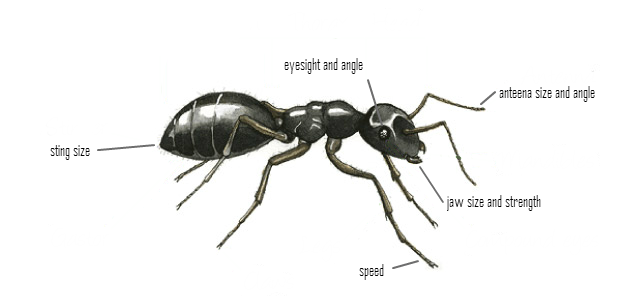
Ant nests are the birth place of all ants. If a nest has excess food i.e. more than enough for it to survive itself, a new ant can be born. The amount of food an ant costs is controlled by the ant’s species. The more food the ant costs, the longer it can survive for after it is born. Ants born from a particular nest remain loyal to that nest e.g. a worker born from a particular nest will only deposit food to that nest.

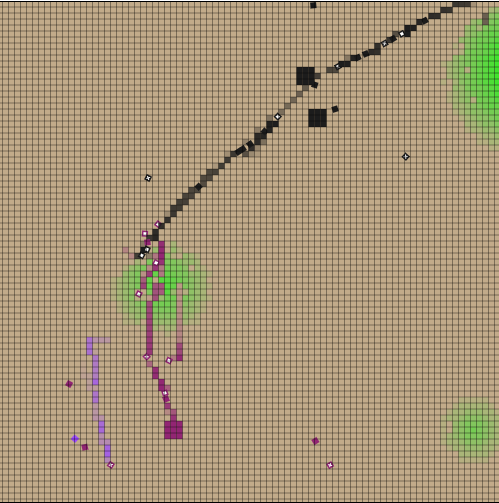


An ant nest

Pheromones are chemical trials deposited by worker ants returning with food back to the nest. They are used by ants that are looking for food, as they should lead to either a source of food or the nest. The stronger a pheromone trial is (the darker its colour) the more likely an ant will follow it. Over time the pheromones concentration will decrease as they evaporate.

Characteristics are features of ants which change their behaviour e.g. the speed characteristic alters how fast an ant can move in the simulation, this is an advantage as it allows the ant to more quickly search for food. However favourable characteristics come at a cost, the more useful a characteristic is the more food is lost in the creation of the ant. So a balance is required to find the most efficient or largest growing population of ants. The characteristics which can be changed are:

* **Speed**- The speed that an ant can move.
* **Jaw Strength** – The strength of the ants jaw (determines how much food the ant can carry).
* **Jaw Size** – The amount of damage a soldier ant can inflict.
* **Sting Size** – The range which a soldier ant can attack.
* **Eyesight –** The range an ant can see.
* **Eye Angle –** The angle through which the ant can see.
* **Antenna Size –** The angle through which an ant can detect pheromones.
* **Pheromone Concentration –** The concentration of pheromones an ant can secrete.
* **Nest Position Memory –** A measure of how well an ant knows where its nest is.
* **Explorative Influence** – The likelihood of an ant changing its direction e.g. not following pheromones.
* **Pheromone Influence** – The likelihood of an ant following a pheromone trial.
* **Worker ant probability** – The probability of a worker ant being born compared with other types of ants.
* **Worker ant food cost** – The amount of food required to create a worker ant (the more health the ant starts with the longer the ant can live before it needs to eat again).
* **Soldier ant probability** – The probability of a soldier ant being born compared with other types of ants.
* **Soldier ant food cost** – The amount of food required to create a soldier ant (the more health the ant starts with the longer the ant can live before it needs to eat again).
* **Queen ant probability** – The probability of a queen ant being born compared with other types of ants.
* **Queen ant food cost** – The amount of food required to create a queen ant (the more health the ant starts with the longer the ant can live before it needs to eat again).
* **Minimum number of Queen steps** – The minimum number of steps a queen will take before reaching its nest site.
* **Maximum number of Queen steps** – The maximum number of steps a queen will take before reaching its nest site.
* **Reproduction rate** – The chance of a new ant being born.

A species is a specific set of values for an ant’s characteristics. There are many species in the simulation, each has a different colour. A species values for characteristics can be edited through the interface however the characteristics of a species have a chance of mutating in the simulation whenever a queen ant is born. If the species does not mutate, the queen will go on to create a nest expanding the spread of the species. However, if a mutation does occur a new species is created and all ants born from the nest which the queen founds will inherit the mutated characteristic. This can be either an advantage or disadvantage on the efficiency of the ant.

Three different species shown, including 3 nests (two of the same species)

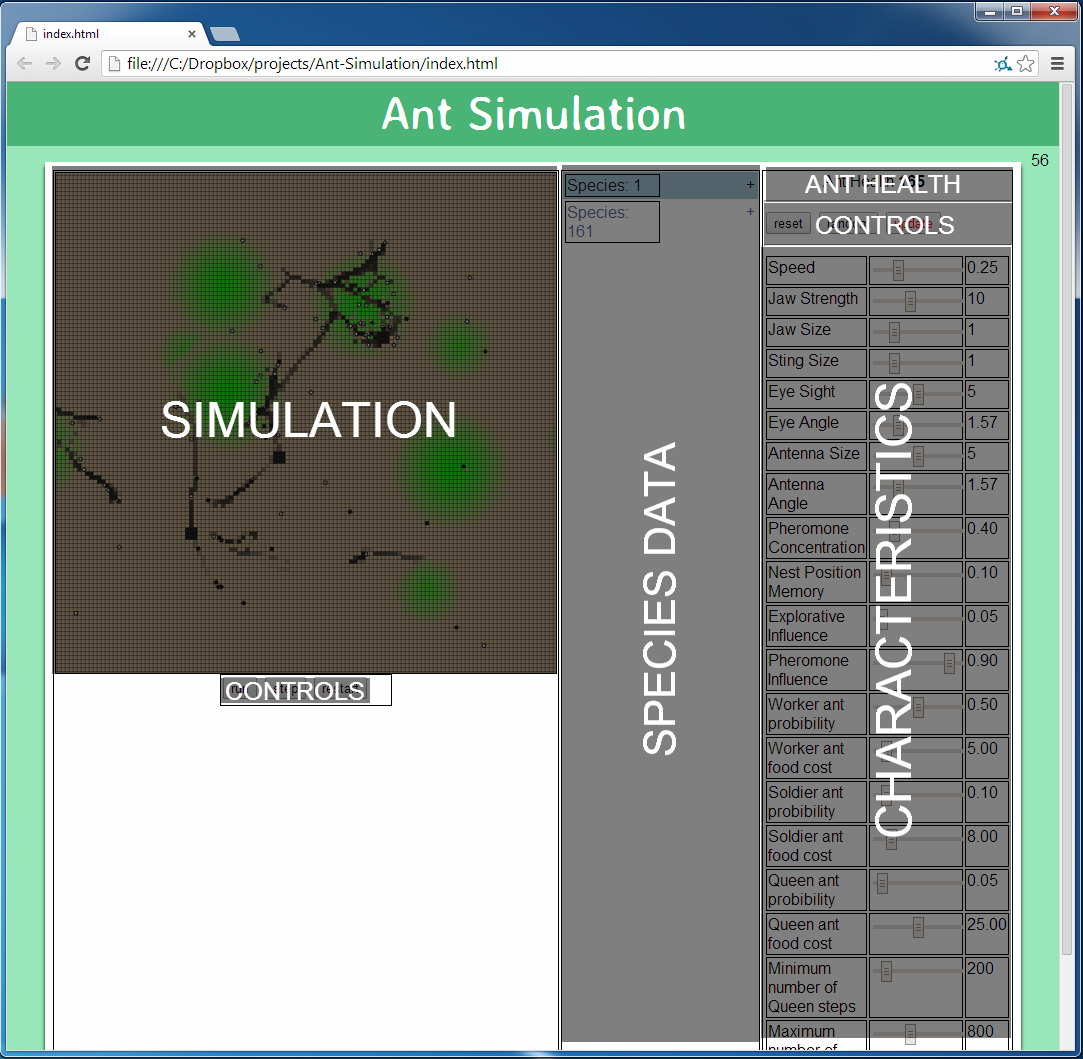
## Installation

There is no need to install the simulation; it can be run via visiting it in a web browser, it will run automatically when the page has loaded.

### System Requirements

|  |  |  |
| --- | --- | --- |
| **Web browser** | **Minimum required** | **Recommended version** |
| Chrome | 4 | 30 or higher |
| Firefox | 4 | 25 or higher |
| Internet Explorer | 10 | 11 or higher |
| Safari | 3.1 | 7.0 or higher |

## Layout

The page is split into three sections, the simulation panel, data panel and configuration panel. 

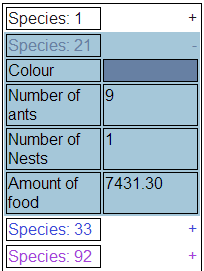
### Simulation panel

Contains the simulation itself, as well as three buttons which are used to control the simulation:

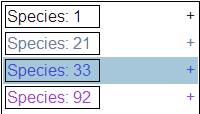
* **Run/Pause** – toggle to start and stop the simulation.
* **Step** – move a single tick forward in the simulation. Useful if you want to closely observe the ant’s behaviour.
* **C:\Dropbox\projects\Ant-Simulation\assests\User Manual\Uses\Time controls.PNGRestart** – Reset the simulation to default settings.

### Data panel

The data panel contains information about all of the species currently in the simulation. As new species are created i.e. mutations occur, they are added to the data panel. Each species can be expanded by clicking on the +/- button to show more information about that species such as the number of ants and nests and the amount of food in the nests. The colour of the species text is the same colour of the species which it represents. You can select a species by clicking on the species name, this will move the simulation to centre on the first nest in the species, it will also display the species characteristics in the configuration panel. (*Note:* the numbers next to the species is the species ID, it does not represent any information about the number of species i.e. Species: 17 is an identifier.)



Species 21 is expanded and selected



The specie panel, all species are contracted. Species: 33 is selected

### Configuration panel

C:\Dropbox\projects\Ant-Simulation\assests\User Manual\Uses\configuration panel controls.PNGThe configuration panel contains the characteristics of the selected species. By changing the inputs, characteristics values can be altered. Once a modification to a characteristic is made the ants health is updated to reflect this change. The buttons at the top control the inputs:



A sample of characteristics

* **Reset** – Resets all characteristics to their default values.
* **Random** – Replaces all characteristics with random values.
* **Update** – Updates the selected species to reflect the changes made in the characteristics panel.

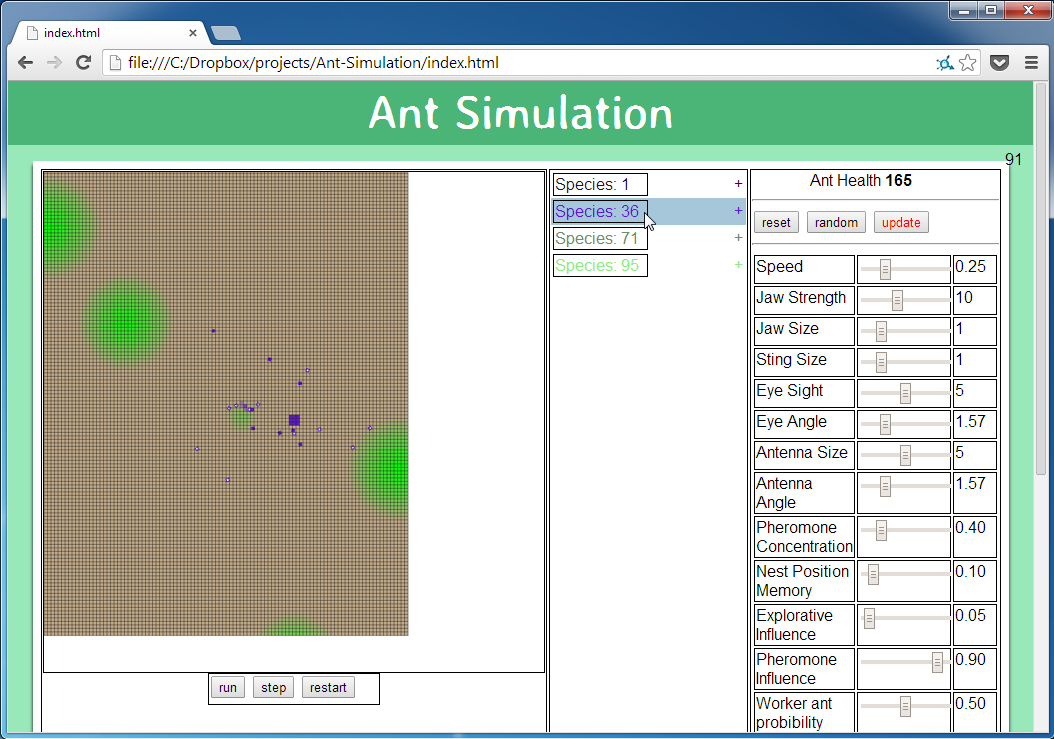
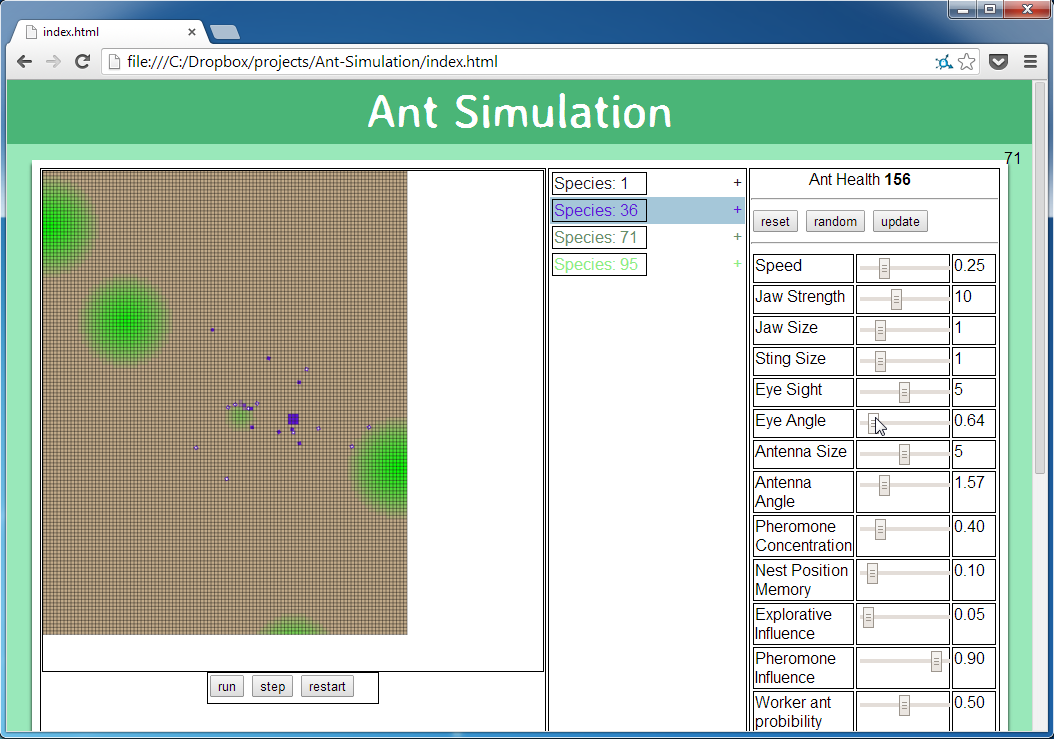
## Use

### How to change characteristics?

To change a characteristic for a particular species:

#### **1** – Select the species

This is done by clicking on the species in the data panel. It will bring up the selected species’ characteristics in the species panel.

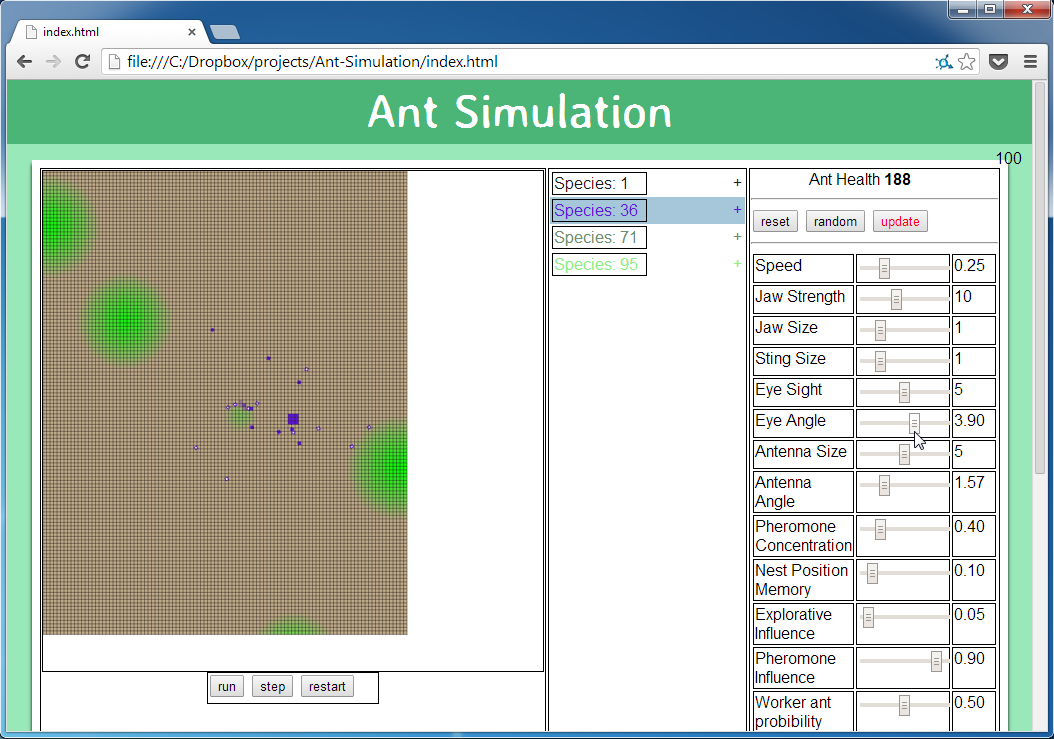


#### **2** – Choose characteristic to change

In the characteristics panel, click on the characteristic you want to change

Note: Hovering over the characteristic will give you a description of what the characteristic does.

#### **3** – Change the characteristic

To change the characteristic move the slider left or right, the value can be seen to the right of the slider for comparison.

Note: This will also change the Ant health and it will change the colour of the update button

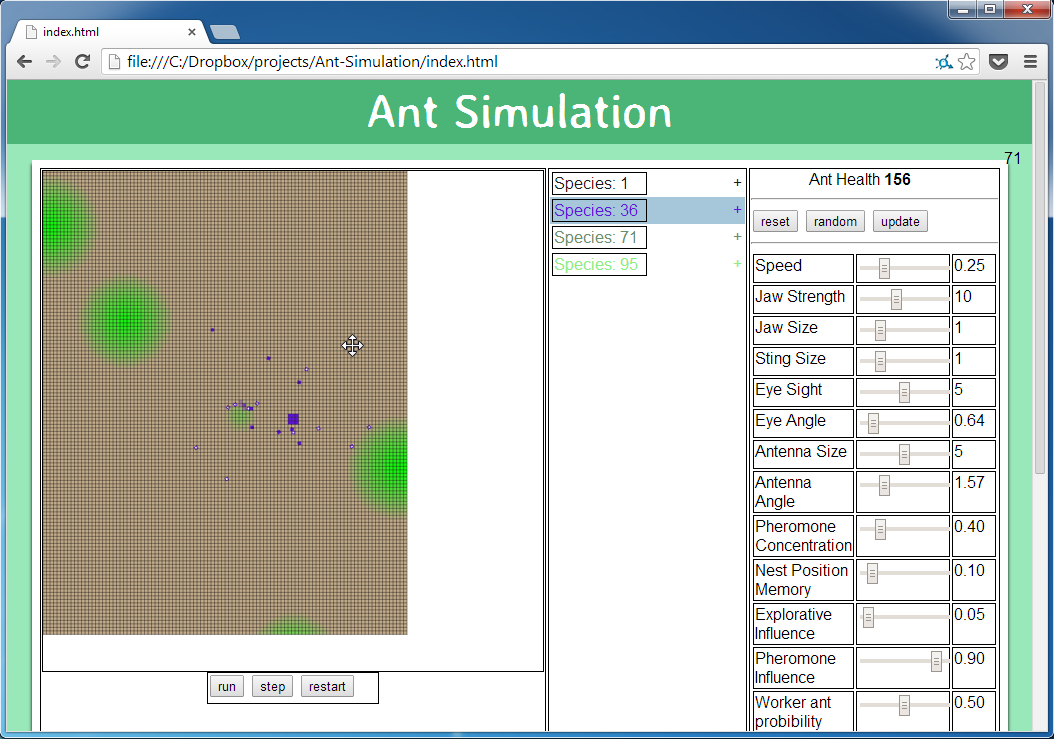
#### **4** – Update

To update the species in the simulation click the update button.

Note: This will change the update button back to its normal colour.

### How to move around the map?

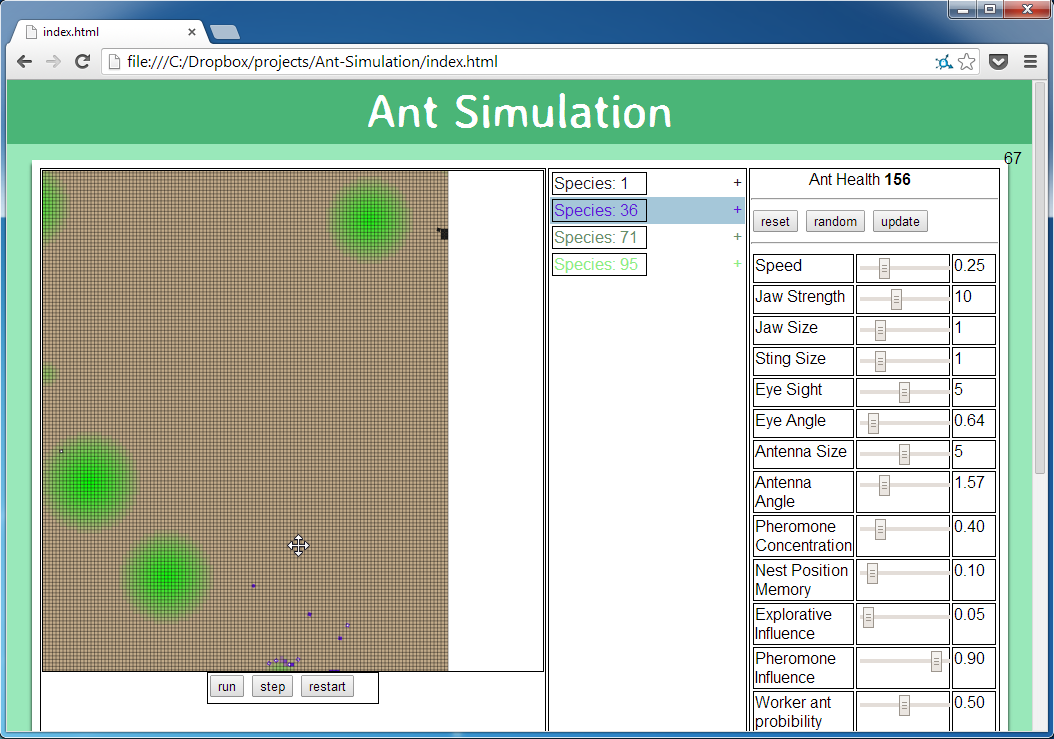
#### **1** – Place cursor over the simulation

Place your cursor anyway over the simulation. The pointer will change to a navigation pointer. 

#### C:\Dropbox\projects\Ant-Simulation\assests\User Manual\Uses\change characteristics\4 - Update.png**2** – Click and drag the cursor

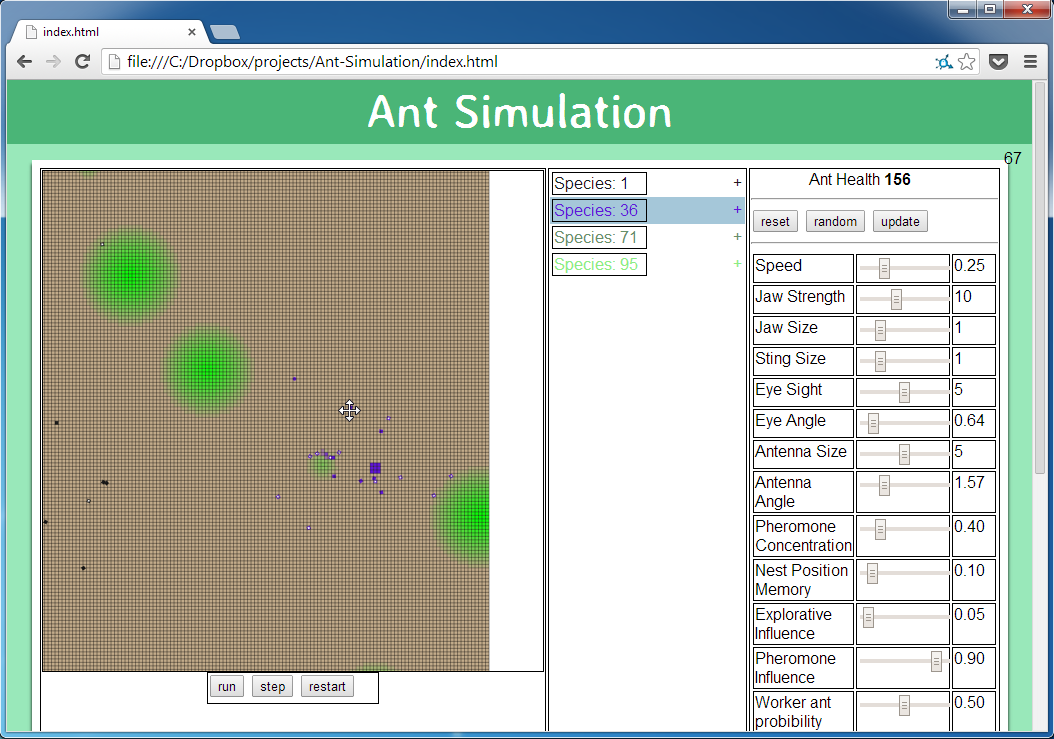
Click and move the mouse, the simulation will follow your mouse.

Note: The arrow keys can also be used to move around the map.

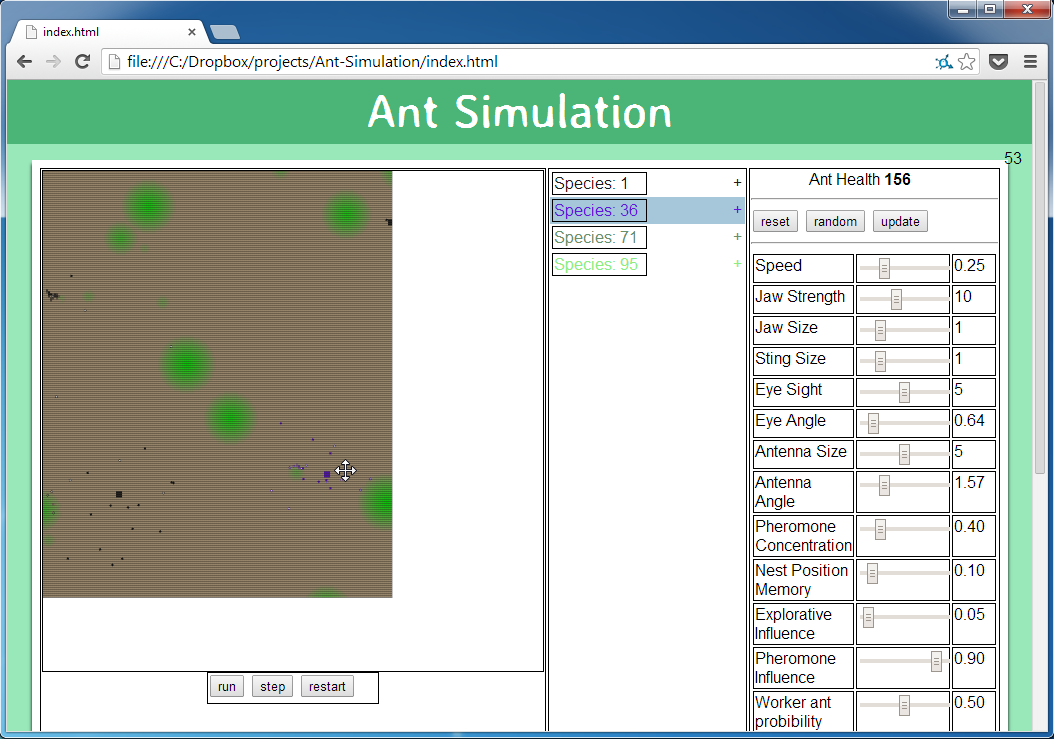


### How to zoom in and out?

#### **1** – Place cursor over the simulation

Place your cursor anyway over the simulation. The pointer will change to a navigation pointer. 

#### **2** – Scroll

Use the mouse scroll wheel up or down to zoom the map in and out respectfully. 

Note: The + and - keys can also be used to zoom in and out.

### Tips and shortcuts

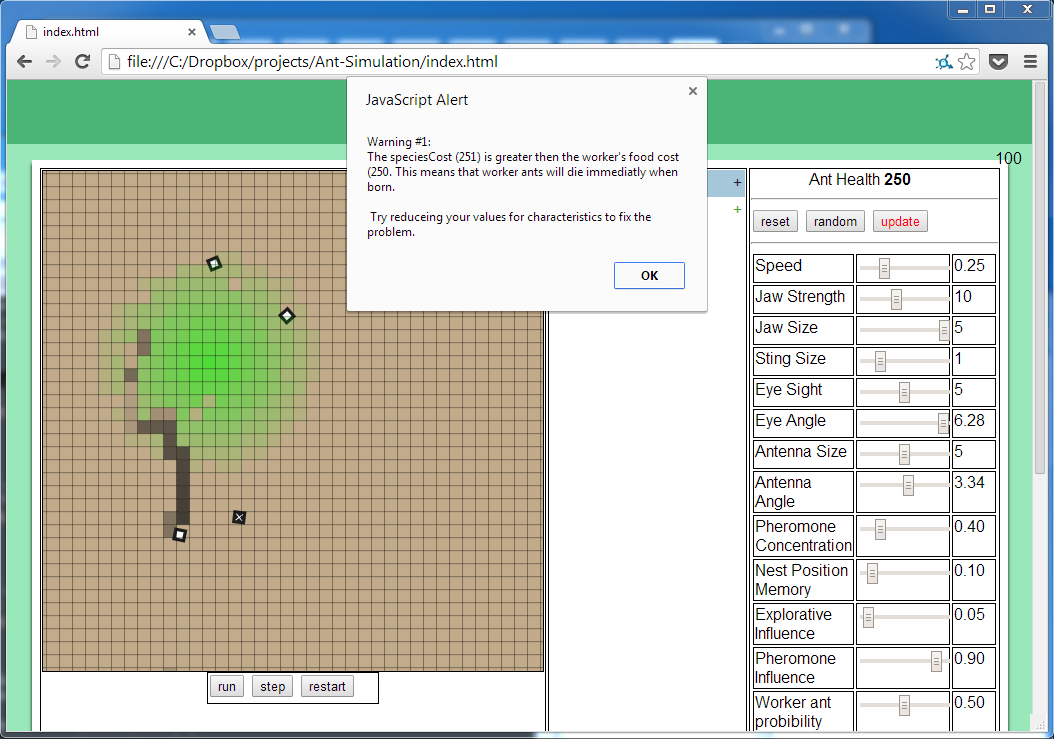
|  |  |
| --- | --- |
| **Control** | **Action** |
| Arrow keys | Pan around the map |
| Mouse click and drag | Pan around the map |
| “+” and “-“ keys | Zoom into and out of the map |
| Mouse scroll | Zoom into and out of the map |
| Space bar | Pause/Play the simulation |
| “r” key | Restart the simulation |
| “s” key | Do a single step in the simulation |

* Hovering over a characteristic will display a description of what it does.
* Clicking on a species will centre the map on the first nest in that species.
* C:\Dropbox\projects\Ant-Simulation\assests\User Manual\Uses\configuration panel controls (update).PNGThe update button will turn red if there is a change to the configuration of a species characteristic which is not reflected in the simulation.

## Errors

### Warning #1 – Species cost greater than workers food cost

#### Problem



The warning is encounter when a species worker ant food cost is lower the ant health. This would cause worker ants to die immediately as they are born.

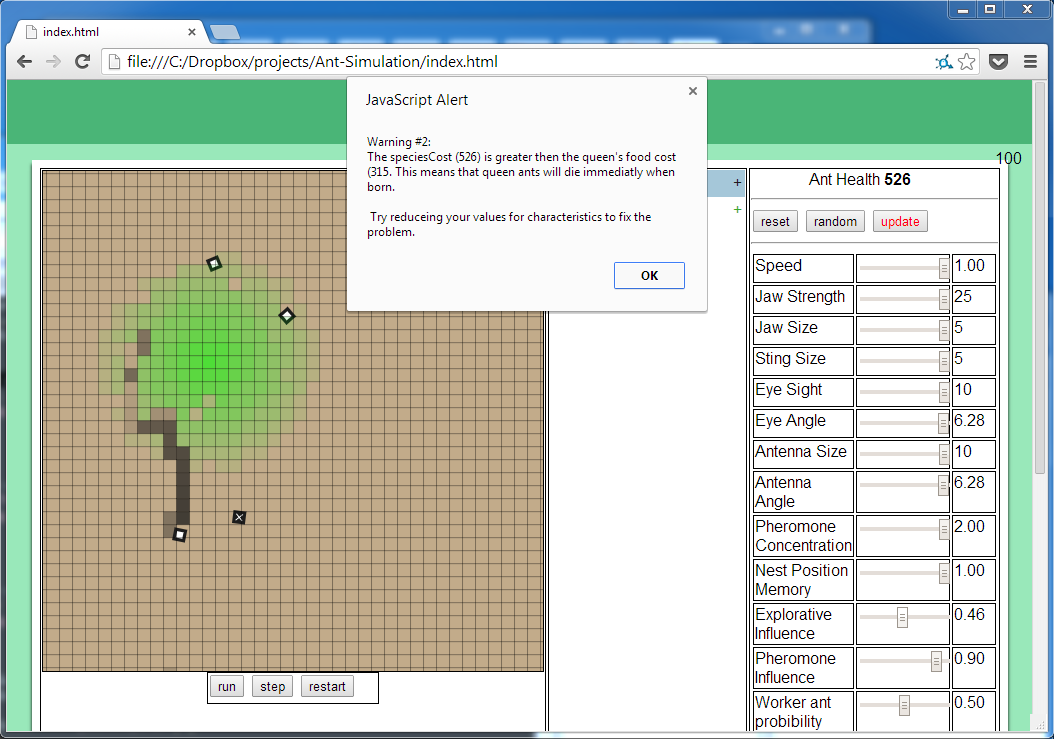
#### Recovery

To recover try either:

* Lowering the species characteristics until ant health is below the worker food cost.
* Raising the worker food cost until it is greater than the ant health.

### Warning #2 – Species cost greater than queen food cost

#### Problem



The warning is encounter when a species queen ant food cost is lower the ant health. This would cause queen ants to die immediately as they are born.

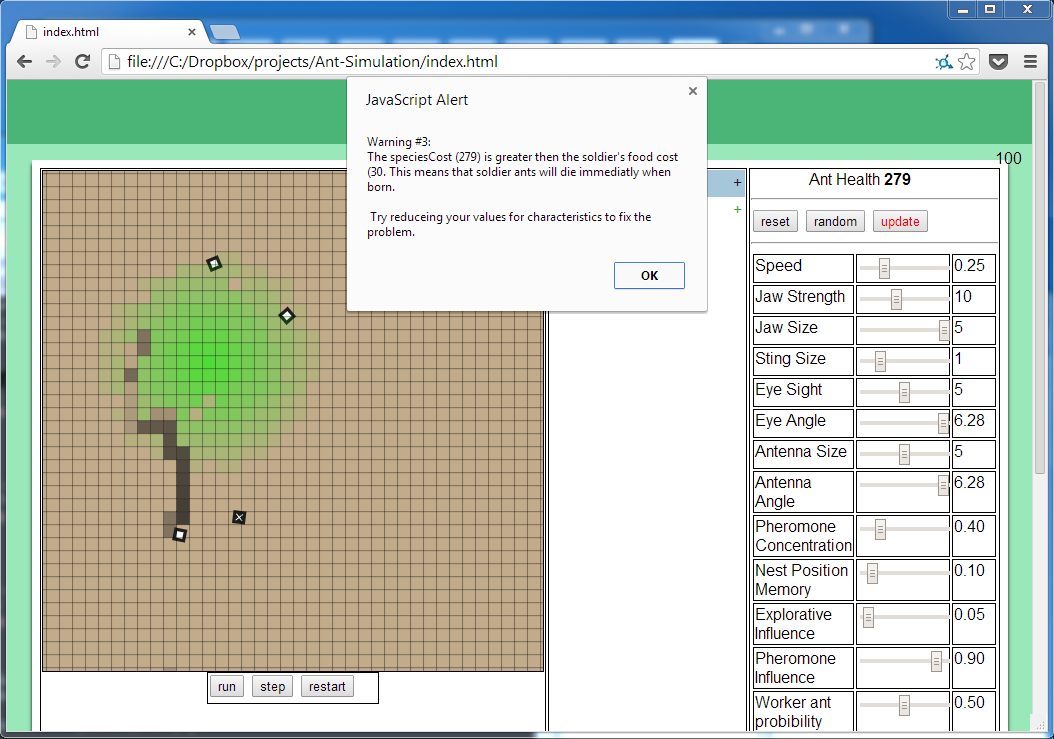
#### Recovery

To recover try either:

* Lowering the species characteristics until ant health is below the queen food cost.
* Raising the queen food cost until it is greater than the ant health.

### Warning #3 – Species cost greater than soldier food cost

#### Problem



The warning is encounter when a species soldier ant food cost is lower the ant health. This would cause soldier ants to die immediately as they are born.

#### Recovery

To recover try either:

* Lowering the species characteristics until ant health is below the soldier food cost.
* Raising the soldier food cost until it is greater than the ant health.

### Error #1 – Minimum number of queen steps greater than maximum

#### Problem



The error is caused when a species minimum number of queen steps is greater than the maximum number of queen steps. This would cause queens to create nests as soon as they were born rather than moving away from the nest.

#### Recovery

To recover from this error try:

* Lowering the minimum number of queen steps until it is below the maximum number of queen steps.
* Raising the maximum number of queen steps until it is greater than the minimum number of queen steps.

### No ants are left

#### Problem

Although not technically a problem, this is caused when all ants and nests have died off.

#### Recovery

The simulation must be restarted. This can be done by clicking the restart button in the simulation controls or by refreshing the page.

### Too many species

#### Problem

This happens when there are many unique species in the simulation and they are all shown in the data panel.

#### Recovery

Suggestions to fix or reduce problem:

* Reduce the queen probability, this will reduce the number of queens produced and therefore reduce the amount of new species.
* Minimise any opened data on species.
* Restart the simulation.

### The simulation is slow

#### Problem

The problem is caused if too many ants are in the simulation. This often happens when the simulation is run for extended amounts of time.

#### Recovery

To reduce the problem try:

* Lowering the worker probability, queen probability and soldier probability characteristics. This will slow the creation of ants in the simulation allowing it to be run for longer periods of time.
* Lowering the eye sight or antenna size characteristics. This will allow the simulation to run faster and handle more ants.
* Lowering the pheromone concentration. This will allow the simulation to run faster as there will be fewer pheromones in the simulation.
* Lowering the reproduction rate. This will reduce the number of ants born. Allowing the simulation to run faster.
* Running the simulation on a faster computer.

### Only one species

#### Problem

The problem is caused if new species are not being created. This happens if queen ants are not being born, or new species nests die quickly after they are created.

#### Recovery

To fix this problem try:

* Increase the reproduction rate. This will allow more ants to be born, increasing the chance of a queen ant being born.
* Increase the queen ant probability. This will cause more queen ants to be born.
* Increase the queen food cost. This will allow other species nests to survive for longer and potentially become successful.
* Design a species to survive for longer by editing the characteristics (try resetting the characteristics by pressing the reset button in the characteristics controls).

### Ants not moving

#### Problem

If ants have a very low speed they will appear not to move

#### Recovery

Increase the speed characteristic of the species.

# Section 7: Evaluation

## Evaluation

How objects where met

### The objectives as set out in the analysis section

#### Simulation objectives

1. A tool which **illustrates the concept of evolution**. This concept is summarised in Appendix B. The main ideas of the concept of evolution should be covered:
   1. **Organisms have characteristics** which determine their species.
   2. As a result of the mutation of characteristics, **variations capable of being inherited** exist within populations of organisms.
   3. Organisms **produce more offspring than can survive**.
   4. These **offspring**, with their different inheritances, **vary in their ability to survive and reproduce**.
   5. In conditions with competition between organisms for survival and reproduction those **organisms with traits that give them an advantage over their competitors pass these advantageous traits on**, while traits that do not confer an advantage are not passed on to the next generation. As a result we have the ‘survival of the fittest’ and a gradual change in populations – they change or may even die out.
2. The simulation should be able to generate **random food placement**.
3. The simulation should be able to simulate basic **growth of food**.
4. The tool should provide a **simulation involving movement** since evolution involves dynamic change and a static tool is not appropriate.
5. The simulation must simulate the three basic types of ants:
   1. The **worker ant** – locates and collects food.
   2. The **soldier ant** – guards and attacks ant species.
   3. The **queen ant** – creates a nest.
6. The simulation must **model a basic nest** capable of:
   1. Producing new ants.
   2. Being attacked and eventually destroyed.
7. The ants should have variable basic characteristics Including:
   1. **Speed** the ants move at.
   2. The **reproduction rate.**
   3. The **amount of food ants are able to carry.**
   4. The ants’ **eyesight** i.e. the distance an ant can see.
   5. The ants’ **antenna size** i.e. the distance the ant can smell pheromones from.
8. The simulation should be able to **introduce random mutations** (so pupils can see how the ants most suited to the environment will survive).
9. The simulation must be able to **show multiple different species at the same time**, so that their fitness can be compared by the pupils.
10. The simulation should model **energy intake from food** needed for an ant to survive depending on the ant’s characteristics. For examples ants with more favourable characteristics such as moving faster should require much more food than ants which have less favourable characteristics.
11. The simulation must be able to model **ants fighting**.
12. The simulation must model **pheromone trails** including:
    1. Their **creation** when an ant is moving.
    2. Their **evaporation** due to conditions.
    3. **How ants respond** to the trails:
       1. Following to find food.
       2. Following back to nest.

#### Simulation interface features objectives

1. **Pause and play buttons** to pause and play simulation.
2. **Navigation buttons** to move around the simulation.
3. **Zooming** in and out of the simulation.
4. For each species (displayed when selecting nest) statistics measuring:
   1. The **amount of food stored in the nest** of a specific species.
   2. **Number of ants** in species.
5. **Editable characteristics** e.g. by sliders or buttons so that the user can change the simulation in real time.
6. **Selection of different species** so that their characteristics can be compared.

### How the objectives where achieved

#### Simulation objectives

1. Evolution is simulated in the following ways
   1. This objective was fully met in the final solution. All ants have characteristics e.g. antenna size. The unique combination of these characteristics determines their species.
   2. When queens are created there is a chance of mutation of a species characteristics. If a mutation occurs a new species is created which has this altered characteristic within it. This new species will then be inherited by all ants which are born from the nest the queen creates. Thus there is a variation between ants born from the original nest and ants born from the nest with the new species.
   3. From multiple runs of the simulation it is clear that not all ants born will survive. Only the ants which are lucky enough to find food will survive for a reasonable amount of time. Thus it can be said that the nests do produce more offspring than can survive.
   4. This is certainly true, not all species flourish in the simulation and in fact many die quickly after being started due to poor mutations in characteristics such as 0 speed which results in ants not being able to move and thus not being able to collect food.
   5. Species in the simulation which have more favourable characteristics will survive for longer than those with poor characteristics due to mutation. The species which survive for longer often produce more queen ants then those which do not survive for as long. As these queen ants will either belong to the same species or belong to a mutate version of the species, the favourable characteristics will be passed onto ants born from the nests these ants make. And from the comment in *d* it is clear that ants with more advantageous characteristics will survive for longer creating a survival of the fittest simulation.
2. At the start or restart of every simulation food is randomly placed throughout the map with random densities. This shows that this objective has been completed.
3. Over time, food will slowly regrow in the simulation (unless it has all been eaten).
4. The simulation allows ants to move on the screen and is certainly not static.
5. The simulation does model simplified versions of the worker, soldier and queen ants. The worker ant can search, collect and deposit food back to the nest. A soldier ant can guard the nest, food and pheromones as well as attack ants from other species. A queen ant can find a location and create a nest. These are sufficient models of the basic ant’s functions.
6. The simulation does model a simple version of an ants nest, the nest in the simulation is effectively a static ant. The nest in the simulation performs the functions of producing new ants and allowing ants to deposit food at it. This is a very simple model of how real ant nests work and could potentially be improved to make the simulation more realistic (*Note*: See suggestions and improvements section for more details). However it is implemented sufficiently enough to say that this objective has been completed.
7. The finished simulation has 21 editable characteristics which all affect the behaviour of the ants in the simulation for example and ant’s eyesight characteristic affects the distance the ant can see in front of it. All characteristics mentioned are impended as well as a number of other characteristics. This objective has been completed beyond expectation.
8. Whenever a queen ant is created in a nest, there is a chance of a single characteristic of the queen’s species to be mutated to a random value. The mutation rate is set high enough for it be quite likely that a queen ant will undergo mutation. This means that there will be a large variation in characteristics from the original characteristics in the simulation thus allowing the user to see which mutations aid ants to survive and which thwart the survival of ants.
9. This objective is complete as multiple different species can be shown, and due to each species having a different colour it is obvious to see which species are fitter than others by the way which they move and act, and by looking at their information in the data panel.
10. Energy intake from food is modelled by the relationship between food and health within the simulation. A single piece of food is worth a certain amount of health. Health is spent by ants in different ways. In all ants health is decreased over time thus creating a reliance on food for survival. Nests have a unique use for health. When an ant is created part of the nests health goes into the ant, this is the amount of health the ant starts with when it is born. However some of the health is lost to the characteristics of the species i.e. species with more favourable characteristics will require more health then less favourable characterises. Therefore there is sufficient evidence to accept that that objective has been met in the simulation.
11. Soldier ants in the simulation can attack other ants. However only other solider ants can fight back. This could be potentially improved by allowing worker and queen ants the ability to fight back or deploy some form of defensive mechanism to stop the soldier ants from attacking *Note*: See the *Improvements* section for more details. However the fighting mechanism is implemented and therefore the objective has been achieved.
12. Pheromones are implemented in the simulation.
    1. Worker ants create pheromone trials when they are returning with food back to their nest thus showing that pheromone trials can be created by moving ants. So this objective has been achieved.
    2. All pheromones in the simulation evaporate at a constant rate i.e. their concentration decreases as time increases until they are gone. The rate can be set in the configuration options and is part of the environmental setup.
    3. All ants except queen ants in the simulation use pheromone trials to navigate. Worker ants will follow pheromone trials in order to either find a source of food, or to find their way back to the nest. Soldier ants also use pheromones trials when they are tasked with guarding pheromone trials, or when they are looking for food to guard.

#### Simulation interface features objectives

1. This objective has been fully met. The simulation can be stopped and started with either a button or a keyboard input. Furthermore a step button has been added which allows the user to step and single tick in the simulation.
2. Navigation has been implemented however its implementation has changed slightly since the design. Instead of buttons overlaying the simulation, mouse input is used instead to navigate and also keyboard input from the arrow keys is used. This implementation is sufficient to say that this objective has been met.
3. Zooming has also been implemented into the interface through either a keyboard or mouse input and thus this objective has been met.
4. As well as both statistics mentioned further statistics have been added such as the number of nests which allow the user to more easily see an overview of a species. This objective has been met.
5. In the interface it is easy for the user to change the characteristics of a species. Sliders have been used rather than buttons to reduce the amount of validation the inputs would have to go through. An update mechanism has been designed to stop the user from accidently changing characteristics; this means that the changes to the simulation are not instantaneous as the user must first press a button however this design decision was made for the better and so the objective has been completed.
6. This has been implemented fully, user can selected different species and view their characteristics.

To summarise all objectives have been fully met, with some objectives being achieved above and beyond they initial ideas stated. Some other objectives although fully implemented could be improved to make the application better.

## User feedback

See *Appendix A – Interviews* for user feedback evidence. An interview was conducted with Dr Sheffrin on Tuesday March 25th to review the completed technical solution.

## Analysis of feedback

*Note*: Not all feedback was documented and some analysis is based on observations of Dr Sheffrin using the application.

### Interface

From feedback interface is clearly laid out and different panels (simulation, data and characteristics) can be easily identified by the user. Hover tool tip information was also obvious to the user from the change in mouse cursor and was useful in helping user understand what characteristics names did. From this feedback it is clear that the design of the interface was well done and is intuitive to the user. This is good as it means that the user will be able to use the application with ease and not get frustrated while using it. It also means that the user will not have to spend much time learning how the interface works as it will be intuitive to them meaning that they will be able to spend more time using the application.

### Simulation Usability

From observations of Dr Sheffrin using the system it was clear that the design goals in making the simulation controls intuitive where achieved. Dr Sheffrin immediately realised how navigation was done by clicking and dragging the mouse and also scrolling to zoom. The use of different cursors help to achive this, as it makes it obvious to the user that there is an action associated with the simulation.

Although the run/pause and the restart buttons where obvious to Dr Sheffrin there was confusion at first about the step button and how it affects time in the simulation. However after experimentation it became clear the function it performs. Dr Sheffrin also commented that the description of the step button in the user manual was clear and that if it was read befor using the simulation there would be no confusion. This is not a big problem in the simulation as the step button will not be a main button used in normal operation of the application.

The representation of the nest was thought to be an ant growing; this is clearly an incorrect idea of what is happening. More distinction is needed between ants and nests to stop this misconception. If nests could grow/be added to so that they become sprawling entities which is more like how they appear in nature (i.e. a series of sprawling tunnels) this could make it clearer. Once this idea is grasped it can be applied to all nests in the simulation. This is a problem with the simulation as a nest is one of the key entities in the simulation. The user manual clearly shows the difference in size between ordinary ants and the nest however maybe an explicit statement that ants in the simulation do not grow could be added in the future.

Despite being discussed in previous interviews the idea of an Ant picture being used to represent an ant was suggested rather than a square. However as discussed in the Design section this would add clutter to the simulation and make it harder to see the behaviour of the ants, although perhaps this should be re-evaluated in the future. A picture of an ant would make it clear to observing users who have not read the user manual or know anything about the application that it simulates ants.

### Characteristics

Some names where cryptic and not immediately obvious to Dr Sheffrin however hover tool tips helped to explain these characteristics well. Although the tool tip which explained the reproduction rate characteristic should be reworded to make it more obvious to the user over which time frame the reproduction probability applies over. The use of sliders made it clear that the characteristics could be altered. The Ant Health field for each ant helped the user understand the relationship between food and health and also how characteristics cost the species health. This is good as it was previousely belived that this was one of the hardest to understand concepts in the simulation.

The default characteristics where commented on (i.e. the characteristics which the simulation starts with and are meant to show a good example of how the simulation runs). The queen ant cost should be raised by x50. This would make queen ants much more rare however would allow nests to create more ants when they are first created which is a better model of real queen ants. This slight alteration could produce much more developed and interesting behaviour much faster than previous defualt settings. This would mean that the user would have less time to wait befor they start to observe some of the evolutionary behaviour in the simulation. This is clearly a good thing as it means less time is wasted waiting for the simulation to get interesting.

The names of the Jaw Strength and Jaw Size characteristics should be swapped. Dr Sheffrin explained how biologically it made more sense for ants with a larger jaw to be able to carry more food while ants with a stronger jaw should do more damage. This does make sense and would help the userability of the application as it would make characteristics more clear to the user.

After altering the characteristics the queen steps min is greater than the queen steps max error appeared. This was not clearly understood and it was suggested that these characteristics should be rolled into one i.e. having a single queen steps characteristics and the application randomly picking values close to this value in a normal distribution. Alternatively a range slider could be implemented with the addition of the JQuery package. An example of a range slider is shown below:

\\fs001\01intake$\01ROBINSON\work\comp\comp4\Ant-Simulation-master(3)\Ant-Simulation-master\writeup\assests\Evaluation\range slider.PNG

This would allow the user to pick a range of values for the queen steps however would eliminate the chance that queen steps min is greater than queen steps max.

### User manual

Dr Sheffrin commented that the user manual was coherent and comprehensive however users could learn much faster by using the application for a few minutes. This shows that the application has been designed effectively and that many features are intuitive or can be easily worked out by users without any training.

### Biological foundation

Dr Sheffrin commented that the simulation was biologically sound and clearly implemented all basic aspects of evolution and demonstrated survival of the fittest. The medium of ants was a good choice to use and the simulation was interesting to watch. It was also mentioned that the concepts demonstrated meant that the simulations use could be expanded throughout the school and be used from years 9 to 13.

### Improvements

The simulation shown used a very large map size and because of this ran slower than deemed acceptable when the simulation was very developed (i.e. 100s of ants). This could be easily fixed by altering the size of the map to make it larger however as discussed in the improvements section a faster update and draw cycle would allow the map to stay this size without compromising the frame rate.

An improvement which was suggested was to have some data which represented the overall success of a species i.e. the number of generations of ants the species has had, the total amount of food collected by a single species. This would allow different simulations to be more easily compared for overall fitness of a species. This would also allow a teacher to ask pupils to design the best species and whoever has the best species (based on length of species life or maybe total amount of food collected) by the end of fifteen minutes would win. This type of application for this tool would be fun and interesting for the students as well as showing the students why certain species died of quickly and why others could live for so long.

To implement this improvement new data would have to be added to the data panel for each species. The potential data could be:

* Length of time species lived
* Total amount of food collected

The length of time a species has lived for can be easily found by keeping track of the tick in which a queen of a new species was born, and then working out the time passed since that moment. The total amount of food collected could also be worked out easily by adding all food collected by worker ants up. These could both be displayed in the data panel for each species easily by modifying the controls module.

## Suggestions and improvements

#### Ants

The simulation could be improved by adding a firmer model of an ant i.e. the current simulation models an ant which borrows features primarily off leaf cutter and Pharaoh ants but has features which do not appear naturally as they are simplifications of much more complex processes such as queens becoming nests. A firmer model would mean that a specific type of ant would be fully implemented rather than a mismatch of multiple types of ants. This would make the simulation easier to conceptualize as it could be more easily compared to real life ants. And this would make the learning process easier to understand for the pupils. This could be done by re analysing how ants work and operate and then modifying the current ant, worker, soldier and queen classes.

Another improvement would be the development of the nest idea in the simulation. Currently the nest is an entity on the map, it acts like a static ant. This is done to reduce the complexity of the simulation. However in reality the nest is a series of tunnels where ants live and work. Expanding the idea of the nest into a location where the queen lives and is tended to by other ants i.e. bringing the queen food, would improve the realism of the simulation. Being able to expand the nest would also be a dramatic improvement and even being able to interconnect multiple nests to create larger nests. This could be achieved by adding the ability for nests to grow over time.

Finally ants in the simulation are born into adulthood, this is a good simplification of the real process however with more time this could be improved on by giving the ants a life cycle i.e. having an infancy stage and then having the ants slowly grow up. The purpose of this would be to improve the realism of the simulation again making its message easier to understand by the pupils as it is more comparable with real ants. This would not be difficult to achieve, it would require ants to have an age property and certain goals/tasks restricted to certain age ranges.

### Pheromones

Only a single pheromone is used in the simulation, the trial pheromone. Most ants in nature have multiple types of pheromones such as attractive pheromones, repulsive pheromones and alert pheromones. A much more real model of the ants search and retrieval of food could be made by integrating these pheromones into the simulation. Due to the abstract nature of the pheromones in the simulation this would be easy to integrate and would involve new pheromone sub classes being created and some logic changes to ants when they smell certain pheromones.

### Species

Of course the number of characteristics in the simulation can always be improved upon. This would make the ants more and more real. This links in with the improvement of a firmer model of the ant. The application has been designed to make it very easy to add and change characteristics. Once added to the CHARS variable an input would be automatically added to the interface. Then they would only need to be added to the species class.

### Faster and larger map

In terms of the technical solution, a faster update cycle would allow for a larger map to be used which would improve the scope of the simulation. It is possible currently to overcrowd the map very quickly by introducing multiple queens very quickly. A larger map would slow down the time it takes for the map to become overcrowded. It would also allow the possibility of more realistic distances between nests. To implement a faster update cycle the code would have to run faster. From current analysis of CPU profiles 25% of the time each tick is spent on the fillRect operation, this is part of the canvas API and can be easily speeded up by doing batch draws rather than drawing individual ants.

### Food

Food in the simulation could be improved upon by adding different types of food which require different characteristics e.g. stomach type, which could allow for carnivores and herbivores which would add another dynamic to the simulation. Food could come in different shapes, this would make it more interesting to watch. This could be done by implementing an algorithm to make randomly shaped food sources perhaps a repurposed map generation algorithm such as the perlin noise algorithm.

# Appendix A: Interviews

The end user is Dr Sheffrin (Head of Sciences and Biology teacher) of Kingswood School, Bath. Alongside two formal interviews, a number of informal interviews have taken place with him, the results of which are summarised below.

*Note*: NMS refers to Dr Sheffrin.

### Tuesday 08/10/13 – Conception

An informal interview was held with Dr Sheffrin where the idea of a simulation to show the aspects of evolution was discussed. The following was asked:

**What are good examples of animals which can be modelled and will be interesting to observe?**

Animals which have large populations are best to model as, having more generations, show a speeded up version of evolution which in animals with smaller populations takes a much longer amount of time and is therefore less interesting to simulate. Insects would probably be the best to simulate.

This was followed by a discussion about either ants or bees as a simulation subject due to their high populations and short life cycles. Each was discussed in depth and ants were chosen due to ants being simpler to understand from a graphical point of view i.e. in 2D ants move along the ground whereas it is difficult to show bees believably in 2D.

### Friday 18/10/13 – Analysis

A formal interview was conducted with Dr Sheffrin after deciding to create an application to help Biology teachers demonstrate the practical aspects of evolution to their pupils in a fun and interesting way. The interview’s objective was to get information to help analyse the problem further.

**What is the aim of the project?**

The aim is to demonstrate the basic principles of evolution in an interesting way, preferably with graphics.

**How is the topic of evolution currently taught?**

It is currently taught in years 8, 11 and in sixth form Biology. Depending on the age group it is covered in various levels of complexity; it is taught over a series of 45 minute lessons of classes of roughly 15 and via homework. The majority of the work is done using worksheets.

**What topics are covered?**

* Mutation
* Natural selection
* Survival of the fittest
* Darwin vs. Lamarckism
* Adaptation
* Habitats and environments
* Characteristics

However for the sixth form topics on DNA and specific process are covered.

**What hardware and software is available?**

During classes an IT room is available with a computer for all of the pupils. There is little software specifically for use of Biology teaching. However there is generic software such as web browsers available on all of the machines.

**How much time is available to use the tool?**

Half of a lesson and a prep.

**How do you wish to use the tool?**

It needs to be used both in the classroom and at home by students on most types of computers. It should be quick to setup and simple to setup and use. There should be a very shallow learning curve to use the tool in order to maximise its effectiveness for everyone.

To make it interesting for the pupils perhaps the pupils should be able to design their own species at the start and can then watch how it mutates over time and how long their species lasts compared with other pupils.

The pupils should be able to step through the simulation to trace how mutated species play out compared with the original species.

**What aspects of evolution should the simulation show?**

The simulation should show how mutations are random events and so do not tend to good or bad adaptations.

The simulation should show how good characteristics often come with negative affects e.g. a large exoskeleton would cause an ant to move more slowly in the environment.

**What should the simulation do?**

The simulation should model a basic environment including regenerating food sources.

The simulation should use ants as the subject through which to show the effects of evolution. Ants should evolve randomly through the species with each species having a set of unique characteristics which define it.

A basic colony structure should be created, maybe including only workers and queens. The whole set of behaviours of the colony are not expected to be modelled.

### Tuesday 05/11/13– Analysis

This informal interview was conducted with Dr Sheffrin to further analyse the potential solution. This interview was mostly focused on ants, their behaviour, characteristics and actions. The following findings where concluded:

* Three main types of ants:
  + **Worker Ants** – which are responsible for searching, collecting and depositing food to the nest.
  + **Queen ants** – which are responsible for generating a new nest as well as reproducing new ants.
  + **Soldier ants** – which are responsible for defending ants of their own species as well as attacking ants from other species.
* Each ant is loyal to its own nest i.e. worker ants will only return food to the nest where they were born.
* All ants can secrete pheromones; there are two main types of pheromones (chemical signals released by ants to share information with other ants):
  + **Alert Pheromones** – these pheromones are released by ants when they are being attacked by other ants. Solder ants from their species will head towards alert pheromones to either attack or defend their species ants. Other types of ants will move away from alert pheromones in order to avoid a battle.
  + **Trial pheromones** – these pheromones are secreted by worker ants which are returning to the nest with food. They are used to show trails to and from the nest to other ants. As a pheromones concentration is collective (i.e. if an ant secretes a pheromones over a patch which already has a pheromone the resulting pheromone will be stronger) it means that the more ants which travel down a path, the more likely that other ants will also travel down the same path.

### Tuesday 14/01/14 – Design

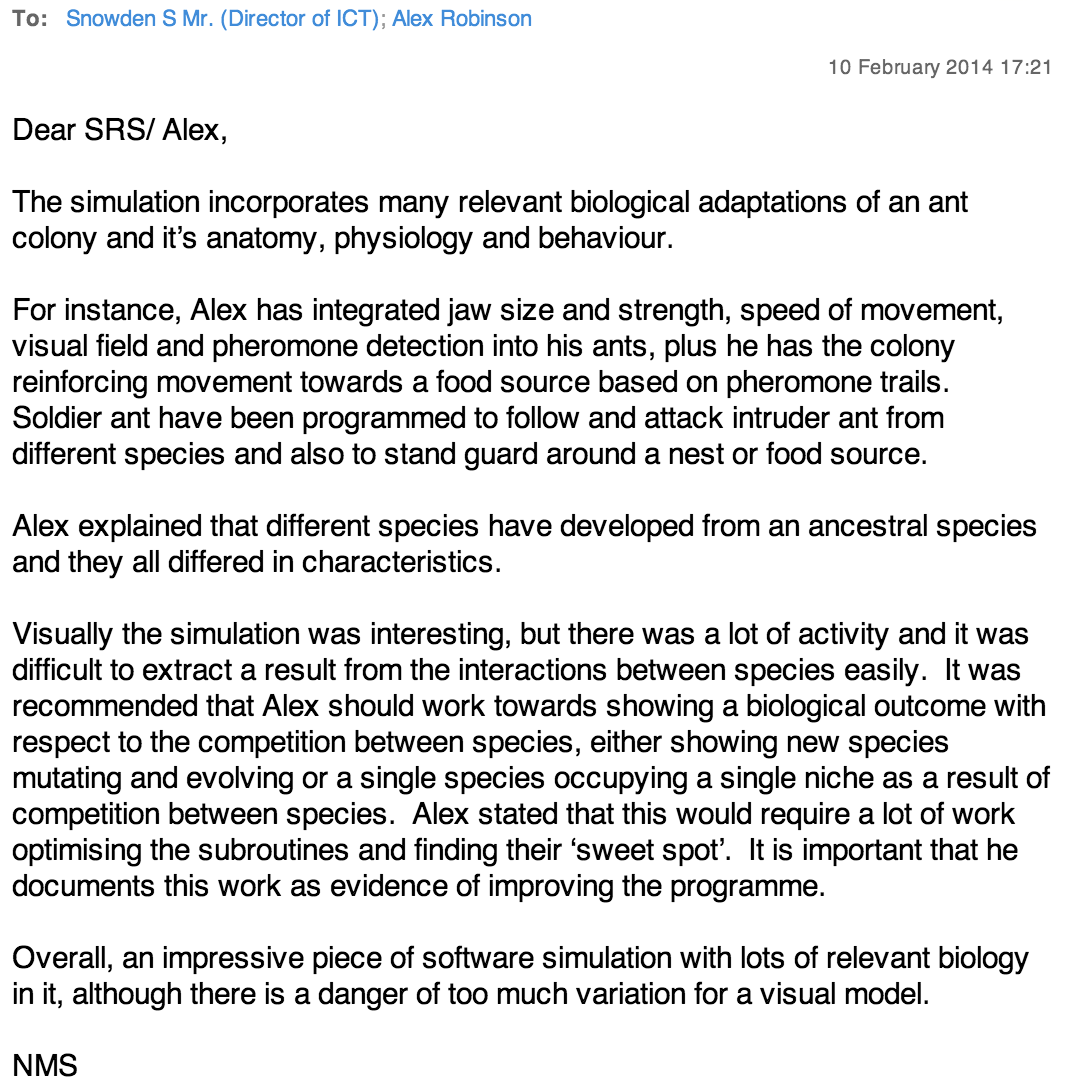
This informal interview with Dr Sheffrin which consisted of various aspects of the design of the application being conceived. Ideas discussed included:

* **Colours** – very bright colours in the simulation, as well as bright colours in the other parts of the interface (however interfaces should not distract the user i.e. they should be drawn to the simulation). Green was discussed as it is often used to represent ideas involving nature, as well as the fact that green is the colour of the school’s Biology department (books, walls, doors…).
* **Structure** – very simple and easy to navigate. Three main sections of the interface were discussed: (1)the simulation itself; (2)the simulation’s output data i.e. the data the user will be able to see about each of the species in the simulation; (3) the input/settings where the user will be able to edit a species’ characteristics and view the selected species’ characteristics.
* **Simulation objects** – very simple shapes should be used to represent objects such as ants, food, nests and pheromones on the map. There is no need to make realistic models of each of these e.g. a picture of a real ant, as this would only make it cluttered and also it would be hard to tell what the image was of if the user was zoomed out the whole way.

### Monday 10/02/14 – Technical review

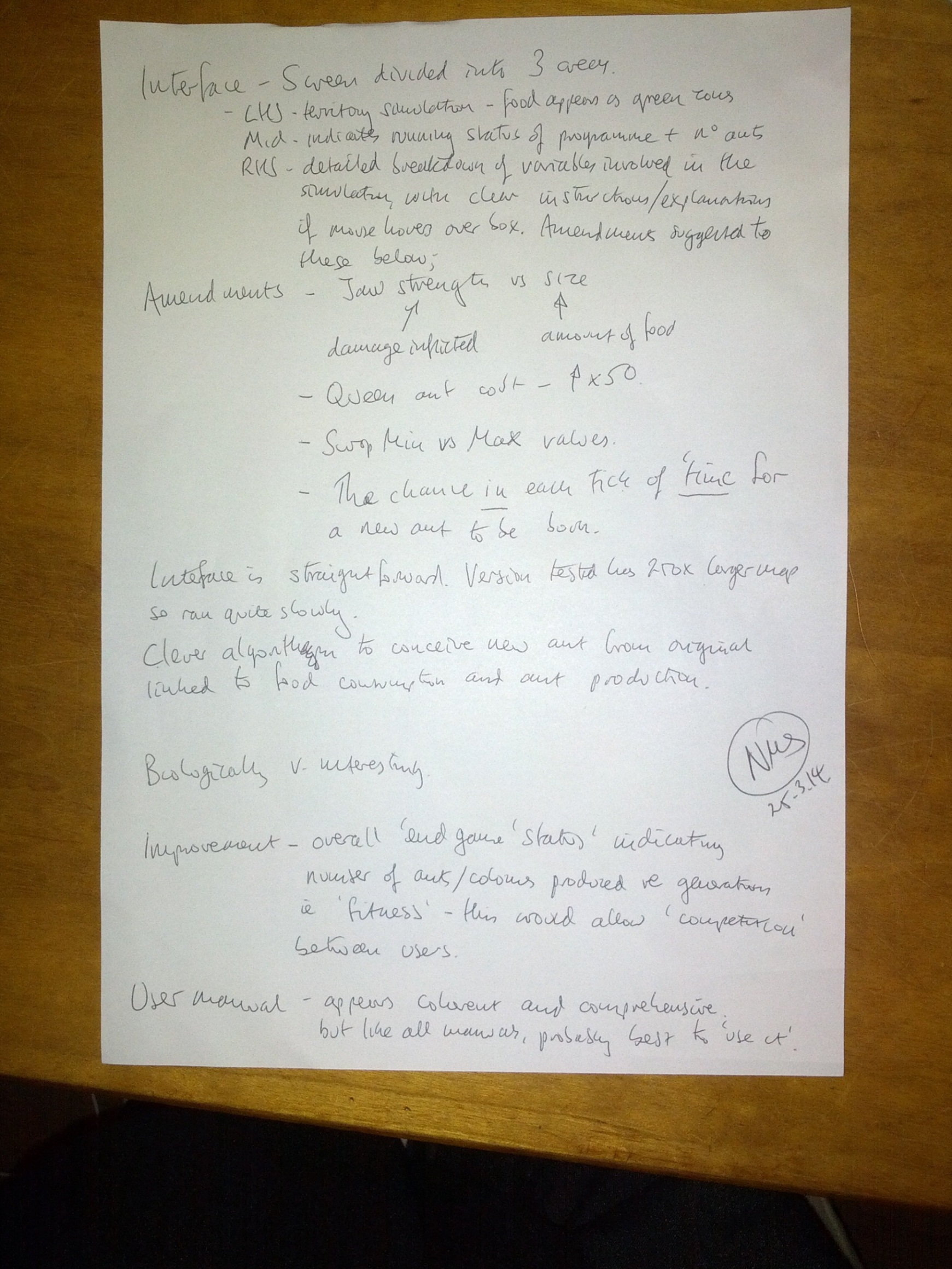
A formal interview took place in which a fully implemented version of the technical solution was shown to Dr Sheffrin. The constructive criticism received included:

* Unclear labelling of characteristics.
* Unclear text summaries of characteristics (in hover text).
* Spelling mistakes in interface.
* Difficult to understand if a single species or multiple species at first.
* Slow progression of dominant species.

The following email of feedback was received:

Tuesday 26/03/14 – Completed project review

A formal interview took place with Dr Sheffrin in which the completed project was shown and the following feedback was received:



# Appendix B: Evolution – The main concepts

From Wikipedia – entry on ‘Evolution’

“Evolution by means of [natural selection](http://en.wikipedia.org/wiki/Natural_selection) is the process by which genetic mutations that enhance reproduction become and remain more common in successive generations of a population. It has often been called a "self-evident" mechanism because it necessarily follows from three simple facts:

* Heritable variation exists within populations of organisms.
* Organisms produce more progeny than can survive.
* These offspring vary in their ability to survive and reproduce.

These conditions produce competition between organisms for survival and reproduction. Consequently, organisms with traits that give them an advantage over their competitors pass these advantageous traits on, while traits that do not confer an advantage are not passed on to the next generation.

The central concept of natural selection is the [evolutionary fitness](http://en.wikipedia.org/wiki/Fitness_%28biology%29) of an organism. Fitness is measured by an organism's ability to survive and reproduce, which determines the size of its genetic contribution to the next generation. However, fitness is not the same as the total number of offspring: instead fitness is indicated by the proportion of subsequent generations that carry an organism's genes. For example, if an organism could survive well and reproduce rapidly, but its offspring were all too small and weak to survive, this organism would make little genetic contribution to future generations and would thus have low fitness.

If an allele increases fitness more than the other alleles of that gene, then with each generation this allele will become more common within the population. These traits are said to be "selected *for*". Examples of traits that can increase fitness are enhanced survival and increased [fecundity](http://en.wikipedia.org/wiki/Fecundity). Conversely, the lower fitness caused by having a less beneficial or deleterious allele results in this allele becoming rarer — they are "selected *against*". Importantly, the fitness of an allele is not a fixed characteristic; if the environment changes, previously neutral or harmful traits may become beneficial and previously beneficial traits become harmful. However, even if the direction of selection does reverse in this way, traits that were lost in the past may not re-evolve in an identical form.”

*Search undertaken 8/10/2013*

# Appendix C: Source code

## index.html

<!DOCTYPE html>

<html>

<head>

<meta content=**"text/html;charset=utf-8"** http-equiv=**"Content-Type"**>

<meta content=**"utf-8"** http-equiv=**"encoding"**>

<title>**Ant Simulation**</title>

<link rel=**"stylesheet"** type=**"text/css"** href=**"style.css"**>

<script src=**"core/config.js"** type=**"text/javascript"**></script>

<script src=**"core/utilities.js"** type=**"text/javascript"**></script>

<script src=**"core/debug.js"** type=**"text/javascript"**></script>

<script src=**"core/canvas.js"** type=**"text/javascript"**></script>

<script src=**"core/map.js"** type=**"text/javascript"**></script>

<script src=**"core/Pheromone.js"** type=**"text/javascript"**></script>

<script src=**"core/Food.js"** type=**"text/javascript"**></script>

<script src=**"core/FoodSystem.js"** type=**"text/javascript"**></script>

<script src=**"core/Species.js"** type=**"text/javascript"**></script>

<script src=**"core/Ant.js"** type=**"text/javascript"**></script>

<script src=**"core/Worker.js"** type=**"text/javascript"**></script>

<script src=**"core/Soldier.js"** type=**"text/javascript"**></script>

<script src=**"core/Queen.js"** type=**"text/javascript"**></script>

<script src=**"core/NestPiece.js"** type=**"text/javascript"**></script>

<script src=**"core/Nest.js"** type=**"text/javascript"**></script>

<script src=**"core/controls.js"** type=**"text/javascript"**></script>

<script src=**"core/main.js"** type=**"text/javascript"**></script>

<script src=**"tests/equal.js"** type=**"text/javascript"**></script>

<script src=**"tests/test.js"** type=**"text/javascript"**></script>

<script src=**"tests/testCase.js"** type=**"text/javascript"**></script>

</head>

<body>

<div id=**'header'**>**Ant Simulation**</div>

<div id=**'page-container'**>

<table>

<tr>

<td id=**'canvas'**>

<div>

<canvas id=**'simulation'** width=**'500px'** height=**'500px'**></canvas>

<span id=**'fps'**>**N/A**</span>

</div>

<div class=**'controls'**>

<button id=**'button-run'** onclick=**'BUTTONS.run(this)'**>**pause**</button>

<button id=**'button-step'** onclick=**'BUTTONS.step(this)'**>**step**</button>

<button id=**'button-restart'** onclick=**'BUTTONS.restart(this)'**>**restart**</button>

</div>

</td>

<td id=**'data'**></td>

<td id=**'config'**>

<div id=**'center'**>

<div class=**'ant-health'**>**Ant Health**</div>

<div class=**'ant-health'** id=**'ant-health'**>**N/A**</div>

</div>

<hr />

<button id=**'button-reset'** onclick=**'BUTTONS.reset()'**>**reset**</button>

<button id=**'button-random'** onclick=**'BUTTONS.random()'**>**random**</button>

<button id=**'button-update'** onclick=**'BUTTONS.update()'**>**update**</button>

<hr />

</td>

</tr>

</table>

</div>

</body>

</html>

## style.css

**@**import url(http://fonts.googleapis.com/css?family=Averia+Sans+Libre|Jolly+Lodger|The+Girl+Next+Door|Wendy+One|Hanalei|Medula+One|Loved+by+the+King);

body **{**

**margin: 0px;**

**padding: 0px;**

**-webkit-user-select: none;**

user-select**: none;**

**background-color: #99E8BA;**

**font-family: Helvetica;**

**}**

**#header** **{**

**background-color: #4AB577;**

**width: 100%;**

**height: 64px;**

**font-size: 3em;**

**font-family:** 'Averia Sans Libre'**, cursive;**

**color: #FFFFFF;**

**text-align: center;**

**}**

**#page-container** **{**

/\* Keeps the body of the page centred with a fixed width \*/

**padding: 6px;**

**width: 964px;**

**margin: auto;**

**margin-top: 16px;**

**margin-bottom: 16px;**

**background-color: #FFFFFF;**

**box-shadow: 0px 4px 4px #000;**

**}**

td **{**

**border: 1px solid black;**

**vertical-align: top;**

**}**

**#fps** **{**

**position: absolute;**

**top: 69px;**

**right: 10px;**

**display: block;**

**}**

canvas **{**

/\* Styles the canvas the simulation runs on \*/

**border: 1px solid black;**

**cursor: move;**

**}**

**.**controls **{**

**width: 170px;**

**height: 30px;**

**border: 1px solid black;**

**margin: auto;**

**}**

td**#data** **{**

**width: 200px;**

**}**

table**.**species **{**

/\* Keeps constant width of data for individual species \*/

**width: 100%;**

**}**

td**.**species **{**

/\* Keeps constant width of rows for species data \*/

**width: 50%;**

**}**

**.**toggleVisibility **{**

**text-align: right;**

**border: none;**

**cursor: pointer;**

**}**

td**#config** **{**

**width: 250px;**

**cursor: help;**

**}**

**#input-container** **{**

/\* Keeps constant width of input \*/

**width: 100%;**

**}**

td**.**config **{**

**width: 1px;**

**}**

input**.**config **{**

/\* Keeps constant width of config\*/

**display: block;**

**width: 100%;**

**}**

div**.**ant-health **{**

**display: inline;**

**}**

**#center** **{**

**width: 130px;**

**margin: auto;**

**}**

**#ant-health** **{**

**font-weight: bold;**

**}**

## core/Ant.js

/\*\*

\* @class Ant

\* @classdesc Represents a single generic ant

\* @param {integer} id - The unique ant id

\* @param {x : number, y : number} coord - The coordinate of the ant

\*/

***var*** Ant **=** ***function*** **(**id**,** coord**)** **{**

/\*\*

\* @property {width : integer, height : integer} this.size - The size of

\* the ant in pixels (default: CELL\_SIZE)

\* @property {x : number, y : number} this.coord - The coordinate of

\* the ant

\* @property {integer} this.id - The unique ant id

\* @property {Species object} this.species - The ants species which

\* determines its characteristics

\* @property {integer} this.type - The type of ant, use ANT\_TYPE object

\* when setting e.g. ANT\_TYPE.worker

\* @property {Nest object} this.nest - The ants home nest where it

\* will deposit food

\* @property {string} this.colour - The hexadecimal colour of the ant

\* @property {number} this.health - The health of the ant, if <= 0 the

\* ant is dead (default: 500)

\* @property {integer} this.hungerThreshold - The value of health

\* bellow which the ant is determined to be hungry (default: 300)

\* @property {number} this.healthRate - The rate at which the ants

\* health decreases per tick

\* @property {boolean} this.alive - (default: true)

\* @property {integer} this.goal - The current goal which the ant is

\* trying to accomplish (default: GOAL.none);

\* @property {x : number, y : number} this.target - The coordinate of a

\* target the ant has choose, type of target depends on ant e.g. worker

\* ants have coordinates of food as their target (default: void(0))

\* @property {ants : [object], food : [object]} this.itemsInView -

\* Arrays of the different items of interest in the ants view

\* (default: {ants : [], food : []})

\* @property {[Pheromone object]} this.pheromonesInRange - An Array of

\* all the pheromones in the ants antenna range (default: [])

\* @property {integer} this.sleep - The number of ticks the ant needs to

\* sleep for. Used for tasks which require actions which take

\* multiple ticks (default: 0)

\* @property {boolean} this.followingPheromone - Used to tell if an ant

\* is following a pheromone (default: false)

\* @property {number} this.direction - The direction in radians from the

\* vertical axis clockwise (default: \*random direction\*)

\* @property {number} this.prioritizeDirection - The direction the ant will

\* general move, used to get straighter more realistic paths

\* (default: \*random direction\*)

\*/

***this*.**size **=** CELL\_SIZE**;**

***this*.**coord **=** coord**;**

***this*.**id **=** id**;**

***this*.**species**;**

***this*.**type**;**

***this*.**nest**;**

***this*.**colour **=** '#1C1C1C'**;**

***this*.**health **=** 0**;**

***this*.**hungerThreshold **=** 100**;**

***this*.**healthRate **=** 0.1**;**

***this*.**alive **=** ***true*;**

***this*.**goal **=** GOAL**.**none**;**

***this*.**target **=** ***void*(**0**);**

***this*.**itemsInView **=** **{**

ants**:** **[],**

food**:** **[]**

**};**

***this*.**pheromonesInRange **=** **[];**

***this*.**sleep **=** 0**;**

***this*.**followingPheromone **=** ***false*;**

***this*.**direction **=** randDir**();**

***this*.**prioritizeDirection **=** randDir**();**

**};**

/\*\*

\* Adds the current position of the ant to the map

\*/

Ant**.*prototype*.**addToMap **=** ***function*** **()** **{**

***if*** **(*this*.**alive**)** MAP**[**coordToIndex**(*this*.**coord**)].**ant**.**push**(*this*);**

**};**

/\*\*

\* Removes the current position of the ant from the map

\*/

Ant**.*prototype*.**removeFromMap **=** ***function*** **()** **{**

***var*** index **=** MAP**[**coordToIndex**(*this*.**coord**)].**ant**.**indexOf**(*this*);**

MAP**[**coordToIndex**(*this*.**coord**)].**ant**.**splice**(**index**,** 1**);**

**};**

/\*\*

\* Determines whether the ant is hungry or not

\* @return {boolean}

\*/

Ant**.*prototype*.**isHungry **=** ***function*** **()** **{**

***if*** **(*this*.**health **<** ***this*.**hungerThreshold**)**

***return*** ***true*;**

***else***

***return*** ***false*;**

**};**

/\*\*

\* Updates the this.sleep variable to simulate time passing during sleep

\*/

Ant**.*prototype*.**updateSleep **=** ***function*** **()** **{**

***if*** **(*this*.**sleep **>** 0**)** **{**

***this*.**sleep **-=** 1**;**

**}**

**};**

/\*\*

\* Determines whether a piece of food exists or not, needed if another ant eats

\* the piece of food being targeted in the same tick

\* @return {boolean}

\*/

Ant**.*prototype*.**isFood **=** ***function*** **(**food**)** **{**

***if*** **(**food **!==** ***void*(**0**)** **&&** food**.**amount **>** 0**)** // food.amount should NEVER be <= 0

***return*** ***true*;**

***else***

***return*** ***false*;**

**};**

/\*\*

\* Takes a single piece of food

\* @return {boolean} - 1 if there is still food, otherwise 0

\*/

Ant**.*prototype*.**takeFood **=** ***function*** **(**food**)** **{**

***if*** **(*this*.**isFood**(**food**))** **{** // If food exists

food**.**amount **-=** 1**;** // Take a single piece of food

***this*.**sleep **+=** ANT\_FOOD\_TAKE\_SPEED**;**

***if*** **(!*this*.**isFood**(**food**))** // If food is all gone remove it from the map

food**.**removeFromMap**();**

***return*** 1**;**

**}** ***else*** **{**

***return*** 0**;**

**}**

**};**

/\*\*

\* Determines whether the ant is currently at its own nest i.e. standing on top

\* of a NestPiece

\* @return {boolean}

\*/

Ant**.*prototype*.**atNest **=** ***function*** **()** **{**

// Check each ant in cell, if any are of type ANT\_TYPE.nest then on nest

***for*** **(*var*** i **=** 0**;** i **<** MAP**[**coordToIndex**(*this*.**coord**)].**ant**.**length**;** i**++)** **{**

***var*** a **=** MAP**[**coordToIndex**(*this*.**coord**)].**ant**[**i**]**

***if*** **(**a**.**type **===** ANT\_TYPE**.**nest **&&** a**.**nest **===** ***this*.**nest**)** **{**

***return*** ***true*;**

**}**

**}**

***return*** ***false*;**

**};**

/\*\*

\* Determines whether the ant can see its own nest

\* @return {boolean}

\*/

Ant**.*prototype*.**seeNest **=** ***function*** **()** **{**

// Checks each ant in view, if any of type ANT\_TYPE.nest, then can see nest

***for*** **(*var*** i **=** 0**;** i **<** ***this*.**itemsInView**.**ants**.**length**;** i**++)** **{**

***var*** a **=** ***this*.**itemsInView**.**ants**[**i**];**

***if*** **(**a**.**type **===** ANT\_TYPE**.**nest **&&** a**.**nest **===** ***this*.**nest**)** **{**

***return*** ***true*;**

**}**

**}**

***return*** ***false*;**

**};**

/\*\*

\* Choose the target piece of food the ant should go for

\*/

Ant**.*prototype*.**findFoodTarget **=** ***function*** **()** **{**

// Ant will pick the first piece of food it sees

***if*** **(*this*.**itemsInView**.**food**.**length **>** 0**)**

***this*.**target **=** ***this*.**itemsInView**.**food**[**0**].**coord**;**

**};**

/\*\*

\* Walk towards food until on top of it and then pick it up

\*/

Ant**.*prototype*.**getFood **=** ***function*** **()** **{**

***this*.**direction **=** angleTo**(*this*.**coord**,** ***this*.**target**);** // Point towards the food

// If on the food pick it up. As this.coord is a number, work out which

// cell the ant is mostly in using coordToIndex

***if*** **(**coordToIndex**(*this*.**coord**)** **===** coordToIndex**(*this*.**target**))** **{**

***this*.**useFood**();**

**}**

**};**

/\*\*

\* Determines the best use of food - Eating it or Carrying it

\*/

Ant**.*prototype*.**useFood **=** ***function*** **()** **{**

***var*** index **=** coordToIndex**(*this*.**coord**);**

***var*** food **=** MAP**[**index**].**food**;**

***if*** **(*this*.**isFood**(**food**))** // check the food still exists i.e. not already eaten

***this*.**health **+=** ***this*.**takeFood**(**food**)** **\*** FOOD\_HEALTH\_RATIO**;**

***else*** // Food is not there/cannot be carried

***this*.**target **=** ***void*(**0**);** // no target

**};**

/\*\*

\* Scans in front of the ant a certain number of cell collecting all items

\* of interest

\*/

Ant**.*prototype*.**scan **=** ***function*** **()** **{**

***this*.**itemsInView **=** **{**

ants**:** **[],**

food**:** **[],**

**};**

// Find the blocks which lie in a sector of a circle of a

// certain radius (eyesight) at a certain angle (eyeAngle) from the ant

// in the direction its facing

***var*** block **=** getSector**(*this*.**coord**,** ***this*.**species**.**chars**.**eyesight**,**

***this*.**direction**,** ***this*.**species**.**chars**.**eyeAngle**);**

// Go through each block and add other ants and pieces of food

***for*** **(*var*** i **=** 0**;** i **<** block**.**length**;** i**++)** **{**

***var*** index **=** coordToIndex**(**block**[**i**]);**

***if*** **(**MAP**[**index**].**ant**.**length **>** 0**)** **{** // Check for ants

***for*** **(*var*** k **=** 0**;** k **<** MAP**[**index**].**ant**.**length**;** k**++)**

***this*.**itemsInView**.**ants**.**push**(**MAP**[**index**].**ant**[**k**]);**

**}**

***if*** **(**MAP**[**index**].**food **!==** ***void*(**0**))** **{** // Check for food

***this*.**itemsInView**.**food**.**push**(**MAP**[**index**].**food**);**

**}**

**}**

**};**

/\*\*

\* Smiler to this.scan() - Scans a certain number of blocks in front at a

\* specific angle and collects all the pheromones

\*/

Ant**.*prototype*.**smell **=** ***function*** **()** **{**

***this*.**pheromonesInRange **=** **[];**

// Find the blocks which lie in a sector of a circle of a

// certain radius (antennaSize) at a certain angle (antennaAngle)

// from the ant in the direction its facing

***var*** block **=** getSector**(*this*.**coord**,** ***this*.**species**.**chars**.**antennaSize**,**

***this*.**direction**,** ***this*.**species**.**chars**.**antennaAngle**);**

// Go through each block and add pheromones to the pheromonesInRange object

***for*** **(*var*** i **=** 0**;** i **<** block**.**length**;** i**++)** **{**

***var*** index **=** coordToIndex**(**block**[**i**]);**

***if*** **(**index **===** coordToIndex**(*this*.**coord**))** // don't smell own square

***continue*;**

***for*** **(*var*** k **=** 0**;** k **<** MAP**[**index**].**pheromone**.**length**;** k**++)** **{**

***this*.**pheromonesInRange**.**push**(**MAP**[**index**].**pheromone**[**k**]);**

**}**

**}**

**};**

/\*\*

\* Secrete pheromones

\*/

Ant**.*prototype*.**secrete **=** ***function*** **()** **{**

***var*** index **=** coordToIndex**(*this*.**coord**);**

***var*** pheromones **=** MAP**[**index**].**pheromone**;**

// Check if there are already pheromones in the cell

***for*** **(*var*** i **=** 0**;** i **<** pheromones**.**length**;** i**++)** **{**

***if*** **(**pheromones**[**i**].**species **==** ***this*.**species**)** **{** // Add to exsisting

pheromones**[**i**].**concentration **+=** ***this*.**species**.**chars**.**pheromoneConcentration**;**

***if*** **(**pheromones**[**i**].**concentration **>** MAX\_PHEROMONE\_CONCENTRATION**)**

pheromones**[**i**].**concentration **=** MAX\_PHEROMONE\_CONCENTRATION**;**

***return*** ***void*(**0**);**

**}**

**}**

// If pheromone from own species not found, create a new pheromone

***var*** pheromone **=** ***new*** Pheromone**(*this*.**species**.**chars**.**pheromoneConcentration**,**

getCellCoord**(*this*.**coord**));**

pheromone**.**species **=** ***this*.**species**;**

pheromone**.**addToMap**();**

**};**

/\*\*

\* Wonder around the map, following pheromone of own species

\*/

Ant**.*prototype*.**wonder **=** ***function*** **()** **{**

***var*** M **=** 0**;**

***var*** Mxy **=** **{**

x**:** 0**,**

y**:** 0

**};**

***var*** CoM **=** **{**

x**:** 0**,**

y**:** 0

**};**

***var*** pheromones **=** ***false*;** // If can follow pheromones i.e. if there are

// no pheromones cannot follow them

// Look at all nearby pheromones of the same species and add up x and y

// coordinates for each as well as the total concentration

***for*** **(*var*** i **=** 0**;** i **<** ***this*.**pheromonesInRange**.**length**;** i**++)** **{**

***if*** **(*this*.**pheromonesInRange**[**i**].**species **===** ***this*.**species**)** **{**

M **+=** ***this*.**pheromonesInRange**[**i**].**concentration**;**

Mxy**.**y **+=** ***this*.**pheromonesInRange**[**i**].**coord**.**y **\*** ***this*.**pheromonesInRange**[**i**].**concentration**;**

Mxy**.**x **+=** ***this*.**pheromonesInRange**[**i**].**coord**.**x **\*** ***this*.**pheromonesInRange**[**i**].**concentration**;**

pheromones **=** ***true*;** // Pheromones to follow

**}**

**}**

CoM **=** **{**

x**:** Mxy**.**x **/** M**,**

y**:** Mxy**.**y **/** M

**};** // The mean coordinate of all the

// pheromones of the same species weighed by pheromone concentration

// Every so often change the prioritize direction

***if*** **(**Math**.**random**()** **<** ***this*.**species**.**chars**.**explorativeInfluence**)**

***this*.**prioritizeDirection **=** randDir**();**

***if*** **(**pheromones **&&** Math**.**random**()** **<** ***this*.**species**.**chars**.**pheromoneInfluence**)** **{**

// If there are pheromones to follow, go towards them however

// there is a chance this will not happen depending on how

// influential pheromones are (pheromoneInfluence)

***var*** angle **=** angleTo**(*this*.**coord**,** CoM**);**

angle **=** validateDirection**(**angle**);**

***if*** **(**angle **>** ***this*.**direction **-** Math**.**PI **/** 3 **&&** angle **<** ***this*.**direction **+** Math**.**PI **/** 3**)** **{**

// will only join pheromones which lie in a cretin angle i.e. simulating

// that ants will not turn directly around

***this*.**direction **=** angle**;**

***this*.**prioritizeDirection **=** ***this*.**direction**;**

***this*.**followingPheromone **=** ***true*;**

**}** ***else*** **{**

***this*.**direction **=** ***this*.**prioritizeDirection**;**

***this*.**followingPheromone **=** ***false*;**

**}**

**}** ***else*** **{** // Otherwise head in the prioritized direction

***this*.**direction **=** ***this*.**prioritizeDirection**;**

***this*.**followingPheromone **=** ***false*;**

**}**

**};**

/\*\*

\* Update the ants coordinates

\*/

Ant**.*prototype*.**move **=** ***function*** **()** **{**

***if*** **(*this*.**sleep **<=** 0**)** **{** // only move when not waiting for a task to complete

***this*.**coord**.**x **+=** Math**.**sin**(*this*.**direction**)** **\*** ***this*.**species**.**chars**.**speed**;**

***this*.**coord**.**y **-=** Math**.**cos**(*this*.**direction**)** **\*** ***this*.**species**.**chars**.**speed**;**

**}**

boundary**(*this*.**coord**,** MAP\_BOUNDARY**);** // Make sure if the ant is out of bounds

**};**

/\*\*

\* Remove the ant from ANT\_LIST and the its species.ants list so the ant is not

\* updated next tick and will be collected by the JS garbage collection

\*/

Ant**.*prototype*.**die **=** ***function*** **()** **{**

***var*** index **=** ANTS\_LIST**.**indexOf**(*this*);**

ANTS\_LIST**.**splice**(**index**,** 1**);** // remove from ANTS\_LIST

***var*** index **=** ***this*.**species**.**ants**.**indexOf**(*this*);** // remove from species.ants

***this*.**species**.**ants**.**splice**(**index**,** 1**);**

***this*.**alive **=** ***false*;**

**};**

## core/Worker.js

/\*\*

\* @class Worker

\* @extends Ant

\* @classdesc Represents a single worker ant

\*/

Worker**.*prototype*** **=** ***new*** Ant**(-**1**,** **{**

x**:** ***void*(**0**),**

y**:** ***void*(**0**)**

**});**

Worker**.*prototype*.**constructor **=** Worker**;**

Worker**.*prototype*.**parent **=** Ant**;**

/\*\*

\* @constructor

\* @param {integer} id - The unique ant id

\* @param {x : number, y : number} coord - The coordinate of the ant

\*/

***function*** Worker**(**id**,** coord**)** **{**

/\*\*

\* @property {integer} this.id - The unique ant id

\* @property {x : number, y : number} this.coord - The coordinate of the ant

\* @property {ANT\_TYPE : integer} this.type - The type of ant i.e. Queen ant

\* (default: ANT\_TYPE.queen)

\* @property {number} this.direction - The direction in radians from the

\* vertical axis clockwise (default: \*random direction\*)

\* @property {number} this.prioritizeDirection - The direction the ant will

\* general move, used to get straighter more realistic paths

\* (default: \*random direction\*)

\* @property {integer} this.carrying - The amount of food the ant is carrying

\* (default: 0)

\* @property {integer} this.carryingThreshold - If an ant is carrying more

\* food then this value and cannot see any food near it,

\* ant will return to the nest to deposit the food (default: 4)

\*/

***this*.**id **=** id**;**

***this*.**coord **=** coord**;**

***this*.**type **=** ANT\_TYPE**.**worker**;**

***this*.**direction **=** randDir**();**

***this*.**prioritizeDirection **=** randDir**();**

***this*.**carrying **=** 0**;**

***this*.**carryingThreshold **=** 0.4**;**

**}**

/\*\*

\* Determines if an ant can carry food or not

\* @return {boolean}

\*/

Worker**.*prototype*.**canCarry **=** ***function*** **()** **{**

***if*** **(*this*.**carrying **<** ***this*.**species**.**chars**.**jawStrength**)**

***return*** ***true*;**

***else***

***return*** ***false*;**

**};**

/\*\*

\* Navigate towards the nest and deposit food at the nest

\*/

Worker**.*prototype*.**depositeFood **=** ***function*** **()** **{**

***if*** **(*this*.**atNest**())** **{** // Deposit food when at the nest

***if*** **(*this*.**dropFood**(*this*.**nest**))** **{**

***this*.**goal **=** GOAL**.**findFood**;**

***this*.**target **=** ***void*(**0**);**

**}**

**}** ***else*** ***if*** **(*this*.**seeNest**())** **{** // If can see the nest, head straight towards it

***this*.**direction **=** angleTo**(*this*.**coord**,** ***this*.**nest**.**coord**);**

**}** ***else*** **{**

***if*** **(**Math**.**random**()** **<** ***this*.**species**.**chars**.**nestCoordMemory**)** // Ant has a sense of the

// direction of the nest

***this*.**prioritizeDirection **=** angleTo**(*this*.**coord**,** ***this*.**nest**.**coord**);**

***this*.**wonder**();** // If not cannot see the nest, walk around randomly

**}**

**};**

/\*\*

\* Drop food at the nest

\* @return {boolean} - True if there is no more food to drop off

\*/

Worker**.*prototype*.**dropFood **=** ***function*** **(**nest**)** **{**

***if*** **(*this*.**carrying **>** 0**)** **{**

nest**.**health **+=** 1 **\*** FOOD\_HEALTH\_RATIO**;**

***this*.**carrying **-=** 1**;**

***this*.**sleep **+=** ANT\_FOOD\_DROP\_SPEED**;**

**}**

***if*** **(*this*.**carrying **>** 0**)** ***return*** ***false*;**

***else*** ***return*** ***true*;**

**};**

/\*\*

\* Determines the best use of food - Eating it or Carrying it

\*/

Worker**.*prototype*.**useFood **=** ***function*** **()** **{**

***var*** index **=** coordToIndex**(*this*.**coord**);**

***var*** food **=** MAP**[**index**].**food**;**

***if*** **(*this*.**hungry **&&** ***this*.**isFood**(**food**))** // Eat the food (this.ifFood(food)

// needed to check the food still exists

// as another ant may have taken it)

***this*.**health **+=** ***this*.**takeFood**(**food**)** **\*** FOOD\_HEALTH\_RATIO**;**

***else*** ***if*** **(*this*.**canCarry**()** **&&** ***this*.**isFood**(**food**))** // Carry the food

***this*.**carrying **+=** ***this*.**takeFood**(**food**);**

***else*** // Food is not there/cannot be carried

***this*.**target **=** ***void*(**0**);**

**};**

/\*\*

\* Performs the actions required to complete a task

\*/

Worker**.*prototype*.**doTask **=** ***function*** **()** **{**

***switch*** **(*this*.**goal**)** **{**

***case*** GOAL**.**findFood**:**

***this*.**wonder**();**

***this*.**findFoodTarget**();**

***break*;**

***case*** GOAL**.**getFood**:**

***this*.**getFood**();**

***break*;**

***case*** GOAL**.**dropFood**:**

***this*.**depositeFood**();**

***this*.**secrete**();**

***break*;**

**}**

**};**

/\*\*

\* Determines if a goal has been completed or not and updates the next goal

\* for the ant

\*/

Worker**.*prototype*.**updateGoal **=** ***function*** **()** **{**

***switch*** **(*this*.**goal**)** **{**

***case*** GOAL**.**none**:**

***this*.**goal **=** GOAL**.**findFood**;**

***break*;**

***case*** GOAL**.**findFood**:**

***if*** **(*this*.**target **!==** ***void*(**0**))** // If found a target

***this*.**goal **=** GOAL**.**getFood**;**

***break*;**

***case*** GOAL**.**getFood**:**

// If have enough food, drop it off at the nest

***if*** **(*this*.**carrying **>=** Math**.**floor**(*this*.**carryingThreshold **\*** ***this*.**species**.**chars**.**jawStrength**))** **{**

***this*.**goal **=** GOAL**.**dropFood**;**

***this*.**target **=** ***void*(**0**);**

**}** ***else*** ***if*** **(*this*.**target **===** ***void*(**0**))** **{** // If the food has been taken by another

// ant, go back to looking for food

***this*.**goal **=** GOAL**.**findFood**;**

**}**

***break*;**

***case*** GOAL**.**dropFood**:**

***this*.**target **=** ***void*(**0**);**

***break*;**

**}**

**};**

Worker**.*prototype*.**updateHealth **=** ***function*** **()** **{**

***if*** **(*this*.**isHungry**()** **&&** ***this*.**carrying **>** 0**)** **{** // If got food and hungry, eat food

***this*.**health **+=** ***this*.**carrying **\*** FOOD\_HEALTH\_RATIO**;**

***this*.**carrying **=** 0**;**

***this*.**goal **=** GOAL**.**none**;**

**}**

***this*.**health **-=** ***this*.**healthRate**;**

***if*** **(*this*.**health **<=** 0**)**

***this*.**die**();**

**};**

/\*\*

\* Draw the ant onto the canvas context

\*/

Worker**.*prototype*.**draw **=** ***function*** **(**ctx**)** **{**

***var*** scaledCoord **=** scaleCoord**(*this*.**coord**);** // Scale the coordinates so they

// map to pixels rather then cells

ctx**.**save**();**

// Translate and rotate the canvas (done so can draw at an angle)

ctx**.**translate**(**scaledCoord**.**x **+** ***this*.**size**.**width **/** 2**,** scaledCoord**.**y **+** ***this*.**size**.**height **/** 2**);**

ctx**.**rotate**(*this*.**direction**);**

drawRect**(**ctx**,** **{**

x**:** **-*this*.**size**.**width **/** 2**,**

y**:** **-*this*.**size**.**height **/** 2

**},** ***this*.**size**,** ***this*.**colour**);**

***if*** **(*this*.**carrying **>** 0**)** **{**

drawRect**(**ctx**,** **{**

x**:** **-*this*.**size**.**width **/** 4**,**

y**:** **-*this*.**size**.**height **/** 4

**},** **{**

width**:** ***this*.**size**.**width **/** 2**,**

height**:** ***this*.**size**.**height **/** 2

**},** FOOD\_COLOUR**);**

**}**

ctx**.**restore**();**

**};**

/\*\*

\* Update the ant each tick

\*/

Worker**.*prototype*.**update **=** ***function*** **()** **{**

***this*.**removeFromMap**();** // Remove from map as ant may move

***this*.**updateHealth**();**

// May have died during the updateHealth so no need to continue if dead

***if*** **(!*this*.**alive**)**

***return*** ***void*(**0**);**

***this*.**scan**();**

***this*.**smell**();**

***this*.**doTask**();**

***this*.**updateGoal**();**

***this*.**updateSleep**();**

***this*.**move**();**

***this*.**addToMap**();** // Once moved add back to map

**};**

## core/Soldier.js

/\*\*

\* @class Soldier

\* @extends Ant

\* @classdesc Represents a single soldier ant

\*/

Soldier**.*prototype*** **=** ***new*** Ant**(-**1**,** **{**

x**:** ***void*(**0**),**

y**:** ***void*(**0**)**

**});**

Soldier**.*prototype*.**constructor **=** Soldier**;**

Soldier**.*prototype*.**parent **=** Ant**;**

/\*\*

\* @constructor

\* @param {integer} id - The unique ant id

\* @param {x : number, y : number} coord - The coordinate of the ant

\*/

***function*** Soldier**(**id**,** coord**)** **{**

/\*\*

\* @property {integer} this.id - The unique ant id

\* @property {x : number, y : number} this.coord - The coordinate of the ant

\* @property {ANT\_TYPE : integer} this.type - The type of ant i.e. Queen ant

\* (default: ANT\_TYPE.queen)

\* @property {number} this.direction - The direction in radians from the

\* vertical axis clockwise (default: \*random direction\*)

\* @property {number} this.prioritizeDirection - The direction the ant will

\* general move, used to get straighter more realistic paths

\* (default: \*random direction\*)

\* @property {integer} this.carrying - The amount of food the ant is carrying

\* (default: 0)

\* @property {integer} this.carryingThreshold - If an ant is carrying more

\* food then this value and cannot see any food near it, ant

\* will return to the nest to deposit the food (default: 4)

\* @property {Ant object} this.targetAnt - The enemy ant which the soldier

\* is targeting to attack (default: void(0))

\* @property {boolean} this.moving - Used for determining when an ant is in

\* a static position i.e. guarding the nest (default:false)

\* @property {integer} this.steps - Used for moving an ant a certain number

\* of steps away from an object. Used for spreading out closely

\* spaced soldier ants (default: \*random integer\*)

\* @property {boolean} this.nearNest - Determines if an ant is close to the

\* nest (even if It cannot see it) (default: false)

\* @property {boolean} this.nearFood - Determines if an ant is close to food

\* (even if It cannot see it) (default: false)

\*/

***this*.**id **=** id**;**

***this*.**coord **=** coord**;**

***this*.**type **=** ANT\_TYPE**.**soldier**;**

***this*.**direction **=** randDir**();**

***this*.**prioritizeDirection **=** randDir**();**

***this*.**targetAnt **=** ***void*(**0**);**

***this*.**moving **=** ***true*;**

***this*.**steps **=** 0**;**

***this*.**nearNest **=** ***false*;**

***this*.**nearFood **=** ***false*;**

**}**

/\*\*

\* Determines if there are other friendly soldiers in view

\* @return {boolean}

\*/

Soldier**.*prototype*.**soldiersInView **=** ***function*** **()** **{**

***for*** **(*var*** i **=** 0**;** i **<** ***this*.**itemsInView**.**ants**.**length**;** i**++)** **{**

***if*** **(*this*.**itemsInView**.**ants**[**i**].**type **===** ANT\_TYPE**.**soldier **&&**

***this*.**itemsInView**.**ants**[**i**].**species **===** ***this*.**species **&&**

***this*.**itemsInView**.**ants**[**i**]** **!==** ***this*)** **{**

***return*** ***true*;**

**}**

**}**

***return*** ***false*;**

**};**

/\*\*

\* Determines if an ant can see food

\* @return {boolean}

\*/

Soldier**.*prototype*.**seeFood **=** ***function*** **()** **{**

***if*** **(*this*.**itemsInView**.**food**.**length **>** 0**)**

***return*** ***true*;**

***else***

***return*** ***false*;**

**};**

/\*\*

\* Chooses which ant the soldier should target

\*/

Soldier**.*prototype*.**pickTarget **=** ***function*** **()** **{**

***if*** **(*this*.**targetAnt **===** ***void*(**0**))** **{** // If don't already have a target

***this*.**targetAnt **=** ***void*(**0**);**

***for*** **(*var*** i **=** 0**;** i **<** ***this*.**itemsInView**.**ants**.**length**;** i**++)** **{**

***var*** ant **=** ***this*.**itemsInView**.**ants**[**i**];**

// If the ant or nest is of a different species

***if*** **(*this*.**species **!==** **((**ant**.**type **===** ANT\_TYPE**.**nest**)** **?** ant**.**nest**.**species **:** ant**.**species**))** **{**

***this*.**targetAnt **=** ant**;** // Set it as the target

***break*;** // Pick first ant in list

**}**

**}**

**}**

**};**

/\*\*

\* Sets the ants direction to intercept the path of the target ant, so that

\* it can attack

\*/

Soldier**.*prototype*.**follow **=** ***function*** **()** **{**

***if*** **(*this*.**targetAnt **!==** ***void*(**0**))** **{**

***this*.**direction **=** angleTo**(*this*.**coord**,** ***this*.**targetAnt**.**coord**);**

***this*.**prioritizeDirection **=** ***this*.**direction**;**

**}**

**};**

/\*\*

\* Attacks a specific ant

\*/

Soldier**.*prototype*.**attack **=** ***function*** **()** **{**

***var*** dist **=** distance**(*this*.**coord**,** ***this*.**targetAnt**.**coord**);**

***if*** **(**dist **<=** ***this*.**species**.**chars**.**stingSize**)** **{** // If the ant is in range

// Do a damage (negative health) to the ant

***this*.**targetAnt**.**health **-=** ***this*.**species**.**chars**.**jawSize **\*** DAMAGE\_MULTIPLIER**;**

***if*** **(*this*.**targetAnt**.**health **<=** 0**)** **{** // If the ant has died

***this*.**targetAnt**.**removeFromMap**();**

***this*.**targetAnt**.**die**();**

***this*.**targetAnt **=** ***void*(**0**);**

**}**

**}** ***else*** ***if*** **(**dist **>** SOLDIER\_ANT\_MAX\_TARGET\_DISTANCE**)** **{** // Stop targeting if far away

***this*.**targetAnt **=** ***void*(**0**);**

**}**

**};**

/\*\*

\* Updates the soldiers ants health

\*/

Soldier**.*prototype*.**updateHealth **=** ***function*** **()** **{**

***this*.**health **-=** ***this*.**healthRate**;**

***if*** **(*this*.**isHungry**())** **{**

***this*.**GOAL **=** GOAL**.**findFood**;**

**}**

***if*** **(*this*.**health **<=** 0**)**

***this*.**die**();**

**};**

/\*\*

\* Updates the this.steps variable

\*/

Soldier**.*prototype*.**updateSteps **=** ***function*** **()** **{**

***if*** **(*this*.**steps **>** 0**)** ***this*.**steps **-=** 1**;**

**};**

/\*\*

\* Controls soldiers logic if assigned goal of guarding the nest

\*/

Soldier**.*prototype*.**guardNest **=** ***function*** **()** **{**

***if*** **(*this*.**nearNest**)** **{** // If the ant is close to the nest

***if*** **(*this*.**steps **<=** 0 **&&** **!*this*.**soldiersInView**())** **{** // and no soldiers in view

***this*.**moving **=** ***false*;**

***this*.**direction **+=** TURN\_RATE**;** // slowly turn i.e. observing surroundings

**}** ***else*** ***if*** **(*this*.**soldiersInView**())** **{** // If soldiers in view, keep moving

***this*.**nearNest **=** ***false*;**

**}**

**}** ***else*** ***if*** **(*this*.**seeNest**()** **&&** **!*this*.**atNest**())** **{** // If near the nest, move a

// specific number of steps away

***this*.**nearNest **=** ***true*;**

***this*.**steps **=** NEST\_GUARD\_RADIUS**;**

**}** ***else*** **{** // Otherwise, keep looking for the nest

***this*.**wonder**();**

***this*.**moving **=** ***true*;**

**}**

**};**

/\*\*

\* Controls soldiers logic if assigned goal of guarding the pheromone trials

\*/

Soldier**.*prototype*.**guardPheromone **=** ***function*** **()** **{**

***this*.**wonder**();** // wonder around following pheromone trials if found

***this*.**moving **=** ***true*;**

**};**

/\*\*

\* Controls soldiers logic if assigned goal of guarding food

\*/

Soldier**.*prototype*.**guardFood **=** ***function*** **()** **{**

***if*** **(*this*.**seeFood**()** **&&** **!*this*.**soldiersInView**())** **{** // If close to food and no other

// soldiers in view

***this*.**moving **=** ***false*;**

***this*.**direction **+=** TURN\_RATE**;** // slowly turn i.e. observing surroundings

**}** ***else*** **{** // Otherwise, keep looking

***this*.**wonder**();**

***this*.**moving **=** ***true*;**

**}**

**};**

/\*\*

\* Controls soldiers logic if it is looking for food

\*/

Soldier**.*prototype*.**findFood **=** ***function*** **()** **{**

***this*.**wonder**();**

***this*.**findFoodTarget**();**

***if*** **(*this*.**target **!==** ***void*(**0**))** **{**

***this*.**getFood**();**

**}**

**};**

/\*\*

\* Performs the actions required to complete a task

\*/

Soldier**.*prototype*.**doTask **=** ***function*** **()** **{**

***switch*** **(*this*.**goal**)** **{**

***case*** GOAL**.**guardNest**:**

***this*.**guardNest**();**

***break*;**

***case*** GOAL**.**guardPheromone**:**

***this*.**guardPheromone**();**

***break*;**

***case*** GOAL**.**guardFood**:**

***this*.**guardFood**();**

***break*;**

***case*** GOAL**.**attack**:**

***this*.**moving **=** ***true*;**

***this*.**follow**();**

***this*.**attack**();**

***break*;**

***case*** GOAL**.**findFood**:**

***this*.**findFood**();**

**}**

**};**

/\*\*

\* Determines if a goal has been completed or not and updates the next goal

\* for the ant

\*/

Soldier**.*prototype*.**updateGoal **=** ***function*** **()** **{**

***switch*** **(*this*.**goal**)** **{**

***case*** GOAL**.**none**:**

***this*.**goal **=** randInt**({**

min**:** GOAL**.**guardNest**,**

max**:** GOAL**.**guardFood

**});**

***break*;**

***case*** GOAL**.**guardNest**:**

***if*** **(*this*.**targetAnt **!==** ***void*(**0**))** // attack interupts task

***this*.**goal **=** GOAL**.**attack**;**

***break*;**

***case*** GOAL**.**guardPheromone**:**

***if*** **(*this*.**targetAnt **!==** ***void*(**0**))**

***this*.**goal **=** GOAL**.**attack**;**

***break*;**

***case*** GOAL**.**guardFood**:**

***if*** **(*this*.**targetAnt **!==** ***void*(**0**))**

***this*.**goal **=** GOAL**.**attack**;**

***break*;**

***case*** GOAL**.**attack**:**

***if*** **(*this*.**targetAnt **===** ***void*(**0**))** // If killed/lost targetAnt, new random goal

***this*.**goal **=** randInt**({**

min**:** GOAL**.**guardNest**,**

max**:** GOAL**.**guardFood

**});**

***break*;**

***case*** GOAL**.**findFood**:**

***if*** **(!*this*.**isHungry**())** // If not hungry choose a random new goal

***this*.**goal **=** randInt**({**

min**:** GOAL**.**guardNest**,**

max**:** GOAL**.**guardFood

**});**

***break*;**

**}**

**};**

/\*\*

\* Draw the ant onto the canvas context

\*/

Soldier**.*prototype*.**draw **=** ***function*** **(**ctx**)** **{**

***var*** scaledCoord **=** scaleCoord**(*this*.**coord**);** // Scale the coordinates so they

// map to pixels rather then cells

ctx**.**save**();**

***var*** crossColour **=** **(*this*.**goal **===** GOAL**.**attack**)** **?** '#FF0000' **:** '#FFFFFF'**;**

// Translate and rotate the canvas (done so can draw at an angle)

ctx**.**translate**(**scaledCoord**.**x **+** ***this*.**size**.**width **/** 2**,** scaledCoord**.**y **+** ***this*.**size**.**height **/** 2**);**

ctx**.**rotate**(*this*.**direction**);**

// Draw the main soldiers body

drawRect**(**ctx**,** **{**

x**:** **-*this*.**size**.**width **/** 2**,**

y**:** **-*this*.**size**.**height **/** 2

**},** ***this*.**size**,** ***this*.**colour**);**

// Draws a cross onto the ant

drawLine**(**ctx**,** **{**

x**:** **-*this*.**size**.**width **/** 4**,**

y**:** **-*this*.**size**.**height **/** 4

**},** **{**

x**:** ***this*.**size**.**width **/** 4**,**

y**:** ***this*.**size**.**height **/** 4

**},** crossColour**,** 1**);**

drawLine**(**ctx**,** **{**

x**:** **-*this*.**size**.**width **/** 4**,**

y**:** ***this*.**size**.**height **/** 4

**},** **{**

x**:** ***this*.**size**.**width **/** 4**,**

y**:** **-*this*.**size**.**height **/** 4

**},** crossColour**,** 1**);**

ctx**.**restore**();**

**};**

/\*\*

\* Update the ant each tick

\*/

Soldier**.*prototype*.**update **=** ***function*** **()** **{**

***this*.**removeFromMap**();** // Remove from map as ant may move

***this*.**updateHealth**();**

// May have died during the updateHealth so no need to continue if dead

***if*** **(!*this*.**alive**)**

***return*** ***void*(**0**);**

***this*.**scan**();**

***this*.**smell**();**

***this*.**pickTarget**();** // Look for potential ants to attack

***this*.**doTask**();**

***this*.**updateGoal**();**

***this*.**updateSleep**();**

***this*.**updateSteps**();**

***if*** **(*this*.**moving**)**

***this*.**move**();**

***this*.**addToMap**();** // Once moved add back to map

**};**

## core/Queen.js

/\*\*

\* @class Queen

\* @extends Ant

\* @classdesc Represents a single queen ant

\*/

Queen**.*prototype*** **=** ***new*** Ant**(-**1**,** **{**

x**:** ***void*(**0**),**

y**:** ***void*(**0**)**

**});**

Queen**.*prototype*.**constructor **=** Queen**;**

Queen**.*prototype*.**parent **=** Ant**;**

/\*\*

\* @constructor

\* @param {integer} id - The unique ant id

\* @param {x : number, y : number} coord - The coordinate of the ant

\*/

***function*** Queen**(**id**,** coord**)** **{**

/\*\*

\* @property {x : number, y : number} this.coord - The coordinate of the ant

\* @property {integer} this.steps - The number of steps the Queen will take

\* until reaching the nest site

\* @property {integer} this.id - The unique ant id

\* @property {GOAL : integer} this.goal - The current goal the ant is trying

\* to accomplish (default: GOAL.none)

\* @property {ANT\_TYPE : integer} this.type - The type of ant i.e. Queen ant

\* (default: ANT\_TYPE.queen)

\*/

***this*.**coord **=** coord**;**

***this*.**steps**;**

***this*.**id **=** id**;**

***this*.**type **=** ANT\_TYPE**.**queen**;**

**}**

/\*\*

\* Decide what actions need to be done to accomplish a task

\*/

Queen**.*prototype*.**doTask **=** ***function*** **()** **{**

***switch*** **(*this*.**goal**)** **{**

***case*** GOAL**.**pickDirection**:**

***this*.**pickDirection**();**

***break*;**

***case*** GOAL**.**gotoNestSite**:**

***this*.**steps **-=** 1**;**

***break*;**

***case*** GOAL**.**createNest**:**

***this*.**createNest**();**

***this*.**die**();**

***break*;**

**}**

**};**

/\*\*

\* Checks to see if the goal is accomplished and updates it if necessary

\*/

Queen**.*prototype*.**updateGoal **=** ***function*** **()** **{**

***switch*** **(*this*.**goal**)** **{**

***case*** GOAL**.**none**:**

***this*.**goal **=** GOAL**.**pickDirection**;**

***break*;**

***case*** GOAL**.**pickDirection**:**

***this*.**goal **=** GOAL**.**gotoNestSite**;**

***break*;**

***case*** GOAL**.**gotoNestSite**:**

***if*** **(*this*.**steps **<=** 0**)** ***this*.**goal **=** GOAL**.**createNest**;**

***break*;**

***case*** GOAL**.**createNest**:**

// Ant should be dead by this point

***break*;**

**}**

**};**

/\*\*

\* Used to pick a direction in which the Queen will walk a specific number of

\* steps in (this.steps) and then create a nest

\*/

Queen**.*prototype*.**pickDirection **=** ***function*** **()** **{**

***this*.**direction **=** randDir**();**

***this*.**steps **=** randInt**({**

min**:** ***this*.**species**.**chars**.**queenStepsMin**,**

max**:** ***this*.**species**.**chars**.**queenStepsMax

**});**

**};**

/\*\*

\* Creates a nest object and deletes the queen

\*/

Queen**.*prototype*.**createNest **=** ***function*** **()** **{**

***var*** nest **=** ***new*** Nest**(**genID**(),** ***this*.**coord**);**

nest**.**species **=** ***this*.**species**;**

nest**.**colour **=** nest**.**species**.**colour**.**nest**;**

nest**.**createNest**();**

nest**.**health **=** ***this*.**health**;**

***var*** index **=** ***this*.**species**.**ants**.**indexOf**(*this*);**

***this*.**species**.**ants**.**splice**(**index**,** 1**);**

nest**.**species**.**nests**.**push**(**nest**);**

ANTS\_LIST**.**push**(**nest**);**

**};**

/\*\*

\* Draw the ant onto the canvas context

\*/

Queen**.*prototype*.**draw **=** ***function*** **(**ctx**)** **{**

***var*** scaledCoord **=** scaleCoord**(*this*.**coord**);** // Scale the coordinates so they

// map to pixels rather then cells

drawCircle**(**ctx**,** scaledCoord**,** ***this*.**size**.**width **/** 2**,** ***this*.**colour**);**

**};**

/\*\*

\* Update the Queen each tick

\*/

Queen**.*prototype*.**update **=** ***function*** **()** **{**

***this*.**removeFromMap**();** // As Queen may move, remove from the map

***this*.**doTask**();**

***this*.**updateGoal**();**

***this*.**move**();**

***this*.**addToMap**();** // Add the Queen back to the map once it has moved

**};**

## core/Species.js

/\*\*

\* @class Species

\* @classdesc Represents an ants species

\* @param {integer} id - The unique ant id

\*/

***var*** Species **=** ***function*** **(**id**)** **{**

/\*\*

\* @property {integer} this.id - The unique ant id

\* @property {[Ant object]} this.ants - List of all ants belonging to

\* the species (used to provide details to the data panel)

\* (default: [])

\* @property {[Nest object]} this.nests - The List of all the nests

\* belonging to the species (used to provide details to

\* the data panel) (default: [])

\* @property {object} this.chars - The species characteristics which

\* influence ants and nests who belong to the species

\* @property {[string]} this.colour - The hexadecimal colours given to

\* each type of ant/nest that belongs to the species

\* @property {number} this.mutationRate - The chance that a characteristic

\* will be mutated

\*/

***this*.**id **=** id**;**

***this*.**ants **=** **[];**

***this*.**nests **=** **[];**

***this*.**chars **=** **{**

/\*\*

\* @property {number} speed - The speed the ant moves, 1 = 1 Cell

\* per tick

\* @property {integer} antennaSize - The range the ant can smell pheromones

\* @property {integer} jawStrength - The strength of the ants jaw

\* (determines how much food the ant can carry)

\* @property {integer} jawSize - The amount of damage a soldier ant

\* does when attack

\* @property {integer} stingSize - The range which the soldier ant

\* can attack

\* @property {integer} eyesight - The range the ant can see items in

\* front of it

\* @property {number} eyeAngle - The angle of the sector the ant can

\* see in front of it

\* @property {number} antennaAngle - The angle of the sector the ant

\* can smell pheromones in front of it

\* @property {number} pheromoneConcentration - The concentration of

\* pheromones secreted

\* @property {number} nestCoordMemory - A measure of how well the ant

\* knows where the nest is, used when navigating to the

\* nest, represents memory of familiarly landmarks

\* near the nest

\* @property {number} explorativeInfluence - The likelihood of an ant

\* changing direction rather then continue going in

\* the direction its facing

\* @property {number} pheromoneInfluence - How likely it is that an

\* ant will follow a pheromones

\* @property {integer} queenStepsMin, queenStepsMax - The range of

\* steps the queen will take when navigating to a new

\* nest site i.e. lower values mean closer nests

\* @property {number} reproductionWorkerProb, reproductionQueenProb, reproductionSoldierProb

\* - The probability a particular type of ant will be

\* born compared with others

\* @property {integer} reproductionWorkerFoodCost, reproductionQueenFoodCost, reproductionSoldierFoodCost

\* - The amount of food required to create this type of

\* ant (This is the amount of food the ant will start with)

\* @property {number} reproductionRate - The chance each tick of

\* creating a new ant

\*/

speed**:** 0.25**,**

antennaSize**:** 5**,**

jawStrength**:** 10**,**

jawSize**:** 1**,**

stingSize**:** 1**,**

eyesight**:** 5**,**

eyeAngle**:** Math**.**PI **/** 2**,**

antennaAngle**:** Math**.**PI **/** 2**,**

pheromoneConcentration**:** 0.4**,**

nestCoordMemory**:** 0.1**,**

explorativeInfluence**:** 0.05**,**

pheromoneInfluence**:** 0.90**,**

queenStepsMin**:** 200**,**

queenStepsMax**:** 800**,**

reproductionWorkerProb**:** 0.5**,**

reproductionWorkerFoodCost**:** 5**,**

reproductionQueenProb**:** 0.05**,**

reproductionQueenFoodCost**:** 25**,**

reproductionSoldierProb**:** 0.1**,**

reproductionSoldierFoodCost**:** 8**,**

reproductionRate**:** 0.05

**};**

***this*.**colour **=** **{**

worker**:** '#1C1C1C'**,**

soldier**:** '#1C1C1C'**,**

queen**:** '#1C1C1C'**,**

nest**:** '#1C1C1C'**,**

pheromone**:** '#1C1C1C'

**};**

***this*.**mutationRate **=** 0.5**;**

**};**

/\*\*

\* Mutate a specific characteristic

\* @param {string} characteristic - The characteristic which will be mutated

\* @return {number} - The mutated value of the characteristic

\*/

Species**.*prototype*.**mutateChar **=** ***function*** **(**characteristic**)** **{**

***var*** charRange **=** CHARS**[**characteristic**];**

// Depending on type of value needed return an random integer or random

// float in a specific range

***if*** **(**charRange**.**type **===** VALUE\_TYPE**.**integerValue**)**

***return*** randInt**(**charRange**);**

***else***

***return*** randFloat**(**charRange**);**

**};**

/\*\*

\* Mutate a single characteristic in the species

\* @return {Species object} - Returns the new mutated species object

\*/

Species**.*prototype*.**mutate **=** ***function*** **()** **{**

***if*** **(**Math**.**random**()** **<=** ***this*.**mutationRate**)** **{**

***var*** altChars **=** clone**(*this*.**chars**);** // Clone the current characteristics

***var*** altCharacteristic **=** randProperty**(**altChars**);** // pick a random characteristic

// to alter

***var*** altValue **=** ***this*.**mutateChar**(**altCharacteristic**);**

altChars**[**altCharacteristic**]** **=** altValue**;**

***var*** species **=** ***this*.**createSpecies**(**altChars**);**

***return*** species**;**

**}** ***else*** **{**

***return*** ***this*;** // Otherwise return an non mutated version

**}**

**};**

/\*\*

\* Creates a new species

\* @param {object} chars - The characteristic set of the new species

\* @return {Species object} - Returns the new species object

\*/

Species**.*prototype*.**createSpecies **=** ***function*** **(**chars**)** **{**

***var*** species **=** ***new*** Species**(**genID**());**

***var*** colour **=** randColour**();**

species**.**chars **=** chars**;**

species**.**colour **=** **{**

worker**:** colour**,**

soldier**:** colour**,**

queen**:** colour**,**

nest**:** colour**,**

pheromone**:** colour**,**

**};**

SPECIES\_LIST**.**push**(**species**);**

createSpeciesData**(**species**);**

***return*** species**;**

**};**

## core/Nest.js

/\*\*

\* @class Nest

\* @classdesc Represents the ant nest entity

\* @param {integer} id - A unique identifier

\* @param {x : number, y : number} coord - The coordinate of the piece

\*/

***var*** Nest **=** ***function*** **(**id**,** coord**)** **{**

/\*\*

\* @property {x : integer, y : integer} this.nestSize - The number of

\* pieces the nest extends both in the x and y directions

\* @property {x : number, y : number} this.coord - The coordinate of the nest

\* @property {integer} this.id - A unique identifier

\* @property {Species object} this.species - The species which the nest belongs to

\* @property {[NestPiece object]} this.pieces - An array of the NestPiece

\* objects belonging to the nest

\* @property {number} this.health - The health the nest has

\* @property {number} this.hungerThreshold - The threshold below which

\* the nest is hungry and tries to preserve food

\* @property {number} this.healthRate - The rate at which the nests health

\* reduces each tick

\* @property {boolean} this.alive - If the nest is alive or not, needed if nest

\* dies mid execution so does not keep acting as if it is alive

\*/

***this*.**nestSize **=** NEST\_SIZE**;**

***this*.**coord **=** coord**;**

***this*.**id **=** id**;**

***this*.**species**;**

***this*.**pieces **=** **[];**

***this*.**health **=** 3000**;**

***this*.**hungerThreshold **=** 300**;**

***this*.**healthRate **=** 1**;**

***this*.**alive **=** ***true*;**

**};**

/\*\*

\* Create a single nest piece

\* @param {x : number, y : number} coord - The coordinate of the nest piece to add

\*/

Nest**.*prototype*.**addNestPiece **=** ***function*** **(**coord**)** **{**

***var*** nestPiece **=** ***new*** NestPiece**(**genID**(),** coord**,** ***this*);**

***this*.**pieces**.**push**(**nestPiece**);**

nestPiece**.**addToMap**();**

**};**

/\*\*

\* Creates all the nest pieces

\*/

Nest**.*prototype*.**createNest **=** ***function*** **()** **{**

***var*** block **=** getBlock**(*this*.**coord**,** ***this*.**nestSize**);** // Get block around nest

***for*** **(*var*** i **=** 0**;** i **<** block**.**length**;** i**++)**

***this*.**addNestPiece**(**block**[**i**]);**

**};**

/\*\*

\* Removing nest from the ants list, it will therefore not be

\* updated and JavaScripts garbage collection should delete the object

\*/

Nest**.*prototype*.**die **=** ***function*** **()** **{**

// Remove from ants list

***var*** index **=** ANTS\_LIST**.**indexOf**(*this*);**

ANTS\_LIST**.**splice**(**index**,** 1**);**

// Remove from species nests list

index **=** ***this*.**species**.**nests**.**indexOf**(*this*);**

***this*.**species**.**nests**.**splice**(**index**,** 1**);**

// Remove from SPECIES\_LIST

***if*** **(*this*.**species**.**nests**.**length **<=** 0**)** **{**

removeSpeciesData**(*this*.**species**.**id**);**

index **=** SPECIES\_LIST**.**indexOf**(*this*.**species**);**

SPECIES\_LIST**.**splice**(**index**,** 1**);**

**}**

// Remove all nest pieces from the map

***for*** **(*var*** i **=** 0**;** i **<** ***this*.**pieces**.**length**;** i**++)** **{**

***this*.**pieces**[**i**].**removeFromMap**();**

**}**

***this*.**alive **=** ***false*;**

**};**

/\*\*

\* Returns the cost in the amount of health needed to create a specific type of ant

\* @param {ANT\_TYPE} type - Represents a specific type of ant

\* @return {number} - The health required

\*/

Nest**.*prototype*.**getCost **=** ***function*** **(**type**)** **{**

***switch*** **(**type**)** **{**

***case*** ANT\_TYPE**.**worker**:**

***return*** ***this*.**species**.**chars**.**reproductionWorkerFoodCost **\*** FOOD\_HEALTH\_RATIO**;**

***case*** ANT\_TYPE**.**queen**:**

***return*** ***this*.**species**.**chars**.**reproductionQueenFoodCost **\*** FOOD\_HEALTH\_RATIO**;**

***case*** ANT\_TYPE**.**soldier**:**

***return*** ***this*.**species**.**chars**.**reproductionSoldierFoodCost **\*** FOOD\_HEALTH\_RATIO**;**

**}**

**};**

/\*\*

\* Calculates the cost of the characteristics due to the species

\* @return {number} - The health cost

\*/

Nest**.*prototype*.**calcSpeciesCost **=** ***function*** **()** **{**

***var*** specieCost **=** 0**;**

***for*** **(**prop ***in*** CHARS**)** **{**

specieCost **+=** CHARS**[**prop**].**healthModifier **\*** ***this*.**species**.**chars**[**prop**];**

**}**

***return*** specieCost**;**

**};**

/\*\*

\* Determines whether or not it is viable to create a specific type of ant

\* @param {ANT\_TYPE} type - Represents a specific type of ant

\* @return {boolean}

\*/

Nest**.*prototype*.**viable **=** ***function*** **(**type**)** **{**

// If creating the ant will make the nest hungry then do not create the ant

***if*** **(*this*.**health **-** **(*this*.**getCost**(**type**)** **+** ***this*.**calcSpeciesCost**())** **>=** ***this*.**hungerThreshold**)**

***return*** ***true*;**

***else***

***return*** ***false*;**

**};**

/\*\*

\* Creates a new ant

\* @param {ANT\_TYPE} type - Represents a specific type of ant

\*/

Nest**.*prototype*.**createAnt **=** ***function*** **(**type**)** **{**

***var*** cost **=** ***this*.**getCost**(**type**);**

createAnt**(*this*.**species**,** **{**

x**:** ***this*.**coord**.**x**,**

y**:** ***this*.**coord**.**y

**},** ***this*,** cost **-** ***this*.**calcSpeciesCost**(),** type**);**

***this*.**health **-=** cost**;**

**};**

/\*\*

\* Determines what type of ant to create

\*/

Nest**.*prototype*.**reproduce **=** ***function*** **()** **{**

***var*** prob **=** Math**.**random**();**

***var*** chars **=** ***this*.**species**.**chars**;**

// Normalize probabilities and then sort into ascending order

***var*** sum **=** chars**.**reproductionQueenProb **+** chars**.**reproductionSoldierProb **+**

chars**.**reproductionWorkerProb**;**

***if*** **(**sum **>** 0**)** **{**

***var*** queenProb **=** chars**.**reproductionQueenProb **/** sum**;**

***var*** soldierProb **=** chars**.**reproductionSoldierProb **/** sum**;**

***var*** workerProb **=** chars**.**reproductionWorkerProb **/** sum**;**

**}**

***var*** ordered **=** **[{**

prob**:** queenProb**,**

type**:** ANT\_TYPE**.**queen

**},** **{**

prob**:** soldierProb**,**

type**:** ANT\_TYPE**.**soldier

**},** **{**

prob**:** workerProb**,**

type**:** ANT\_TYPE**.**worker

**}].**sort**(*function*** **(**a**,** b**)** **{**

***return*** a**.**prob **-** b**.**prob**;**

**});** // sort min to max

// Determine which outcome occurred

***if*** **(**prob **<** ordered**[**0**].**prob **&&** ***this*.**viable**(**ordered**[**0**].**type**))**

***this*.**createAnt**(**ordered**[**0**].**type**);**

***else*** ***if*** **((**prob **<** ordered**[**1**].**prob **+** ordered**[**0**].**prob**)** **&&** // cumulative probability

***this*.**viable**(**ordered**[**1**].**type**))**

***this*.**createAnt**(**ordered**[**1**].**type**);**

***else*** ***if*** **(**prob **<** ordered**[**2**].**prob **+** ordered**[**1**].**prob **+** ordered**[**0**].**prob **&&**

***this*.**viable**(**ordered**[**2**].**type**))**

***this*.**createAnt**(**ordered**[**2**].**type**);**

**};**

/\*\*

\* Updates nests health

\*/

Nest**.*prototype*.**updateHealth **=** ***function*** **()** **{**

***this*.**health **-=** ***this*.**healthRate**;**

***if*** **(*this*.**health **<=** 0**)**

***this*.**die**();**

**};**

/\*\*

\* Updates the nest each tick

\*/

Nest**.*prototype*.**update **=** ***function*** **()** **{**

***this*.**updateHealth**();**

***if*** **(!*this*.**alive**)**

***return*** ***void*(**0**);**

// Determines if a new ant should be created or not

***if*** **(**Math**.**random**()** **<** ***this*.**species**.**chars**.**reproductionRate**)** **{**

***this*.**reproduce**();**

**}**

**};**

## core/NestPiece.js

/\*\*

\* @class NestPiece

\* @classdesc Represents a single piece of the nest

\* @param {integer} id - A unique identifier

\* @param {x : number, y : number} coord - The coordinate of the piece

\* @param {Nest object} nest - The parent nest class which controls the nest piece

\*/

***var*** NestPiece **=** ***function*** **(**id**,** coord**,** nest**)** **{**

/\*\*

\* @property {width : number, height : number} this.size - The size the

\* piece will be in pixels

\* @property {x : number, y : number} this.coord - The coordinate of the piece

\* @property {Nest object} this.nest - The parent nest class which controls

\* the nest piece

\* @property {integer} this.id - A unique identifier

\* @property {integer} this.type - The type of object the piece is. As the

\* nest piece is essentially treated as a static ant and

\* stored under ant in MAP, the type is needed so ants

\* can identify the piece

\* @property {number} this.health - The health the piece has

\*/

***this*.**size **=** CELL\_SIZE**;**

***this*.**coord **=** coord**;**

***this*.**nest **=** nest**;**

***this*.**id **=** id**;**

***this*.**type **=** ANT\_TYPE**.**nest**;**

***this*.**health **=** ***this*.**nest**.**health**;**

**};**

/\*\*

\* Adds the piece to the map

\*/

NestPiece**.*prototype*.**addToMap **=** ***function*** **()** **{**

***if*** **(*this*.**nest**.**alive**)**

MAP**[**coordToIndex**(*this*.**coord**)].**ant**.**push**(*this*);**

**};**

/\*\*

\* Removes the piece from the map

\*/

NestPiece**.*prototype*.**removeFromMap **=** ***function*** **()** **{**

***var*** index **=** MAP**[**coordToIndex**(*this*.**coord**)].**ant**.**indexOf**(*this*);**

MAP**[**coordToIndex**(*this*.**coord**)].**ant**.**splice**(**index**,** 1**);**

**};**

/\*\*

\* Removes the piece from the map

\*/

NestPiece**.*prototype*.**die **=** ***function*** **()** **{**

***var*** index **=** ***this*.**nest**.**pieces**.**indexOf**(*this*);**

***this*.**nest**.**pieces**.**splice**(**index**,** 1**);**

***if*** **(*this*.**nest**.**pieces**.**length **<=** 0**)** **{**

***this*.**nest**.**die**();**

**}**

**};**

/\*\*

\* Draws the piece onto the canvas context

\* @param {canvas context 2d} ctx - The context which the piece will be drawn onto

\*/

NestPiece**.*prototype*.**draw **=** ***function*** **(**ctx**)** **{**

drawRect**(**ctx**,** scaleCoord**(*this*.**coord**),** ***this*.**size**,** ***this*.**nest**.**species**.**colour**.**nest**);**

**};**

## core/Pheromone.js

/\*\*

\* @class Pheromone

\* @classdesc Represents a single pheromone

\* @param {float} concentration - The concentration of the pheromone

\* @param {x : number, y : number} coord - The coordinate of the pheromone

\*/

***var*** Pheromone **=** ***function*** **(**concentration**,** coord**)** **{**

/\*\*

\* @property {float} this.concentration - The concentration of the pheromone

\* @property {x : number, y : number} this.coord - The coordinate of

\* the pheromone

\* @property {width : number, height : number} this.size - The size the

\* pheromone will be in pixels

\* @property {Species object} this.species - The species which the pheromone

\* belongs to

\*/

***this*.**concentration **=** concentration**;**

***this*.**coord **=** coord**;**

***this*.**size **=** CELL\_SIZE**;**

***this*.**species**;**

**};**

/\*\*

\* Adds the pheromone to the map

\*/

Pheromone**.*prototype*.**addToMap **=** ***function*** **()** **{**

MAP**[**coordToIndex**(*this*.**coord**)].**pheromone**.**push**(*this*);**

**};**

/\*\*

\* Removes the pheromone from the map

\*/

Pheromone**.*prototype*.**removeFromMap **=** ***function*** **()** **{**

***var*** index **=** MAP**[**coordToIndex**(*this*.**coord**)].**pheromone**.**indexOf**(*this*);**

MAP**[**coordToIndex**(*this*.**coord**)].**pheromone**.**splice**(**index**,** 1**);**

**};**

/\*\*

\* Draws the pheromone onto the canvas context

\* @param {canvas context 2d} ctx - The context which the pheromone will be drawn onto

\*/

Pheromone**.*prototype*.**draw **=** ***function*** **(**ctx**)** **{**

***if*** **(*this*.**concentration **>** 0**)** **{** // If pheromone exists

ctx**.**globalAlpha **=** ***this*.**concentration**;** // alter opacity depending on pheromone concentration

drawRect**(**ctx**,** scaleCoord**(*this*.**coord**),** ***this*.**size**,** ***this*.**species**.**colour**.**pheromone**);**

ctx**.**globalAlpha **=** 1**;** // Reset opacity

**}**

**};**

/\*\*

\* Update the pheromones concentration each tick

\*/

Pheromone**.*prototype*.**update **=** ***function*** **()** **{**

***this*.**concentration **-=** PHEROMONE\_EVAPERATION\_RATE**;**

// Remove pheromone if no concentration

***if*** **(*this*.**concentration **<=** 0**)**

***this*.**removeFromMap**();**

**};**

## core/FoodSystem.js

/\*\*

\* @class FoodSystem

\* @classdesc Controls all the food on the map

\*/

***var*** FoodSystem **=** ***function*** **()** **{**

/\*\*

\* @property {min : integer, max : integer} this.variation - The minimum

\* and maximum amounts of food a single piece can contain

\* @property {string} this.colour - The colour of the food

\*/

***this*.**variation **=** **{**

min**:** 0**,**

max**:** 15

**};**

***this*.**colour **=** FOOD\_COLOUR**;**

**};**

/\*\*

\* Creates a circle of food with reducing amounts around a coordinate

\* @param {x : number, y : number} coord - The coordinate of the centre of the circle

\* @param {integer} radius - The maximum distance a piece of food can be from

\* the centre coord

\*/

FoodSystem**.*prototype*.**addFoodBlob **=** ***function*** **(**coord**,** radius**)** **{**

***var*** affectedCells **=** getSector**(**coord**,** radius**,** 0**,** 2 **\*** Math**.**PI**);** // All cells in a circle

// For each cell, calculate the amount it should have

***for*** **(*var*** i **=** 0**;** i **<** affectedCells**.**length**;** i**++)** **{**

***var*** distanceFromCenter **=** distance**(**coord**,** affectedCells**[**i**]);**

***var*** amount **=** radius **-** Math**.**round**(**distanceFromCenter**);**

***if*** **(**amount **>=** ***this*.**variation**.**min**)** // create food and add to map

MAP**[**coordToIndex**(**affectedCells**[**i**])].**food **=** ***new*** Food**(*this*,** amount**,** affectedCells**[**i**]);**

**}**

**};**

/\*\*

\* Adds random amounts of food at random positions in the map

\*/

FoodSystem**.*prototype*.**addFood **=** ***function*** **()** **{**

***for*** **(*var*** i **=** 0**;** i **<** NUM\_OF\_CELLS**;** i**++)** **{**

***if*** **(**Math**.**random**()** **<** FOOD\_CHANCE**)** **{**

***this*.**addFoodBlob**(**indexToCoord**(**i**),** randInt**(*this*.**variation**));**

**}**

**}**

**};**

/\*\*

\* Randomly picks pieces of food in the map to grow. The probability of a single

\* piece of food being selected is set in config.js

\*/

FoodSystem**.*prototype*.**growFood **=** ***function*** **()** **{**

***if*** **(**Math**.**random**()** **<** FOOD\_GROW\_RATE**)** **{**

***for*** **(*var*** i **=** 0**;** i **<** FOOD\_GROW\_AMOUNT**;** i**++)** **{**

// Pick a random index

***var*** index **=** randInt**({**

min**:** 0**,**

max**:** NUM\_OF\_CELLS **-** 1

**});**

***var*** food **=** MAP**[**index**].**food**;**

// Determine if it has food

***if*** **(**food **!==** ***void*(**0**))** **{**

food**.**grow**();**

**}** ***else*** **{**

***continue*;**

**}**

**}**

**}**

**}**

## core/Food.js

/\*\*

\* @class Food

\* @classdesc Represents a single piece of food

\* @param {FoodSystem object} foodSystem - A parent food system

\* @param {integer} amount - The concentration of food the piece has

\* @param {x : number, y : number} coord - The coordinate of the piece of food

\*/

***var*** Food **=** ***function*** **(**foodSystem**,** amount**,** coord**)** **{**

/\*\*

\* @property {FoodSystem object} this.foodSystem - The food system which

\* controls the food pieces

\* @property {width : number, height : number} this.size - The size in pixels

\* @property {integer} this.amount - The concentration of food the piece has

\* @property {x : number, y : number} this.coord - The coordinate of the

\* piece of food

\*/

***this*.**foodSystem **=** foodSystem**;**

***this*.**size **=** CELL\_SIZE**;**

***this*.**amount **=** amount**;**

***this*.**coord **=** coord**;**

**};**

/\*\*

\* Adds the piece of food to the map

\*/

Food**.*prototype*.**addToMap **=** ***function*** **()** **{**

***var*** index **=** coordToIndex**(*this*.**coord**);**

MAP**[**index**].**food **=** ***this*;**

**};**

/\*\*

\* Removes the piece of food from the map

\*/

Food**.*prototype*.**removeFromMap **=** ***function*** **()** **{**

***var*** index **=** coordToIndex**(*this*.**coord**);** // Legal as only one piece of food per cell

MAP**[**index**].**food **=** ***void*(**0**);**

**};**

/\*\*

\* Draws the piece of food onto the canvas context

\* @param {canvas context 2d} ctx - The context which the food will be drawn onto

\*/

Food**.*prototype*.**draw **=** ***function*** **(**ctx**)** **{**

ctx**.**globalAlpha **=** ***this*.**amount **/** ***this*.**foodSystem**.**variation**.**max**;** // alter opacity

drawRect**(**ctx**,** scaleCoord**(*this*.**coord**),** ***this*.**size**,** ***this*.**foodSystem**.**colour**);**

ctx**.**globalAlpha **=** 1**;** // reset opacity

**};**

/\*\*

\* Adds to the amount of all surrounding pieces of food. Creates surrounding

\* food if it dose not exist.

\*/

Food**.*prototype*.**grow **=** ***function*** **()** **{**

***this*.**amount**++;** // adds 1 amount to itself

// Gets block around food

***var*** block **=** getBlock**(*this*.**coord**,** **{**

width**:** 1**,**

height**:** 1

**});**

// For each piece of food, if already exists add 1 to amount otherwise

// create new piece of food.

***for*** **(*var*** i **=** 0**;** i **<** block**.**length**;** i**++)** **{**

***var*** food **=** MAP**[**coordToIndex**(**block**[**i**])].**food**;**

***if*** **(**food **!==** ***void*(**0**))** **{**

food**.**amount**++;**

**}** ***else*** **{**

MAP**[**coordToIndex**(**block**[**i**])].**food **=** ***new*** Food**(*this*.**foodSystem**,** 1**,** block**[**i**]);**

**}**

**}**

**}**

## core/map.js

/\*\*

\* Draws a grid onto the canvas context

\* @param {canvas context 2d} ctx - The context which the grid will be drawn onto

\*/

***function*** drawGrid**(**ctx**)** **{**

***for*** **(*var*** x **=** 0**;** x **<** GRID\_SIZE**.**width **+** 1**;** x **+=** 1**)** // Draw lines parallel to the x-axis

drawLine**(**ctx**,** **{**

x**:** x **\*** CELL\_SIZE**.**width **+** CANVAS\_OFFSET**.**x**,**

y**:** CANVAS\_OFFSET**.**y

**},** **{**

x**:** x **\*** CELL\_SIZE**.**width **+** CANVAS\_OFFSET**.**x**,**

y**:** GRID\_SIZE**.**height **\*** CELL\_SIZE**.**height **+** CANVAS\_OFFSET**.**y

**},** GRID\_COLOUR**,** GRID\_LINE\_WIDTH**);**

***for*** **(*var*** y **=** 0**;** y **<** GRID\_SIZE**.**height **+** 1**;** y **+=** 1**)** // Draw lines parallel to the y-axis

drawLine**(**ctx**,** **{**

x**:** CANVAS\_OFFSET**.**x**,**

y**:** y **\*** CELL\_SIZE**.**height **+** CANVAS\_OFFSET**.**y

**},** **{**

x**:** CELL\_SIZE**.**height **\*** GRID\_SIZE**.**height **+** CANVAS\_OFFSET**.**x**,**

y**:** y **\*** CELL\_SIZE**.**height **+** CANVAS\_OFFSET**.**y

**},** GRID\_COLOUR**,** GRID\_LINE\_WIDTH**);**

**}**

/\*\*

\* Draws a rectangle over the entire canvas effectively wiping it to a single colour

\* @param {canvas context 2d} ctx - The context which the rectangle will be drawn onto

\*/

***function*** drawBackground**(**ctx**)** **{**

drawRect**(**ctx**,** CANVAS\_OFFSET**,** **{**

width**:** GRID\_SIZE**.**width **\*** CELL\_SIZE**.**width**,**

height**:** GRID\_SIZE**.**height **\*** CELL\_SIZE**.**height

**},** BACKGROUND\_COLOUR**);**

**}**

/\*\*

\* Change simulation zoom level

\* @param {number} level - The zoom level (positive : zoom in, negative : zoom out)

\*/

***function*** zoom**(**level**)** **{**

***if*** **(**CELL\_SIZE**.**width **+** level **>** MIN\_ZOOM **&&** CELL\_SIZE**.**width **+** level **<** MAX\_ZOOM **&&**

CELL\_SIZE**.**height **+** level **>** MIN\_ZOOM **&&** CELL\_SIZE**.**height **+** level **<** MAX\_ZOOM**)** **{**

CELL\_SIZE**.**width **+=** level**;**

CELL\_SIZE**.**height **+=** level**;**

**}**

**}**

/\*\*

\* Create an empty map and populate it with default values

\*/

***function*** createMap**()** **{**

***for*** **(*var*** i **=** 0**;** i **<** NUM\_OF\_CELLS**;** i**++)**

MAP**[**i**]** **=** **{**

ant**:** **[],**

food**:** ***void*(**0**),**

pheromone**:** **[]**

**};**

**}**

## core/controls.js

***function*** start**()** **{**

RUNNING **=** ***true*;**

getElement**(**'button-run'**).**innerHTML **=** 'pause'**;**

**}**

***function*** pause**()** **{**

RUNNING **=** ***false*;**

getElement**(**'button-run'**).**innerHTML **=** 'run'**;**

**}**

***function*** toggleRunning**()** **{**

***if*** **(**RUNNING**)** pause**();**

***else*** start**();**

**}**

/\*\*

\* Steps through a single tick

\*/

***function*** step**()** **{**

start**();**

tick**();**

pause**();**

**}**

/\*\*

\* Toggles RUNNING and updates the pause/run button

\*/

***function*** runPauseButton**()** **{**

toggleRunning**();**

***if*** **(**RUNNING**)**

getElement**(**'button-run'**).**innerHTML **=** 'pause'**;**

***else***

getElement**(**'button-run'**).**innerHTML **=** 'run'**;**

**}**

/\*\*

\* Updates the value of a characteristic

\* @param {HTML element} element - An element of a input in the settings panel

\* @param {\*} value - The new value of the characteristic

\*/

***function*** updateValue**(**element**,** value**)** **{**

// Update the button colour to show an update needs to be pushed to the species

getElement**(**'button-update'**).**style**.**color **=** BUTTON\_UPDATE\_COLOUR**;**

// Get the characteristic the element referees to and set its to the new value

***var*** characteristic **=** CHARS**[**element**.**name**];**

characteristic**.**value **=** value**;**

***if*** **(**characteristic**.**type **===** VALUE\_TYPE**.**floatValue**)**

value **=** parseFloat**(**value**).**toFixed**(**NUMBER\_OF\_FIXED\_PLACES**);**

// Re calculate the species cost

***var*** specieCost **=** 0**;**

***for*** **(**prop ***in*** CHARS**)** **{**

specieCost **+=** CHARS**[**prop**].**healthModifier **\*** CHARS**[**prop**].**value**;**

**}**

***var*** workerFoodCost **=** CHARS**.**reproductionWorkerFoodCost**.**value **\*** FOOD\_HEALTH\_RATIO**;**

***var*** queenFoodCost **=** CHARS**.**reproductionQueenFoodCost**.**value **\*** FOOD\_HEALTH\_RATIO**;**

***var*** soldierFoodCost **=** CHARS**.**reproductionSoldierFoodCost**.**value **\*** FOOD\_HEALTH\_RATIO**;**

***if*** **((**workerFoodCost **-** specieCost**)** **<** 0**)** **{**

console**.**log**(**'YELL'**)**

window**.**alert**(**'Warning #1:\nThe speciesCost (' **+** specieCost**.**toFixed**(**0**)** **+** ') is greater then \

the worker\'s food cost (' **+** workerFoodCost **+** '. This means that worker ants \

will die immediatly when born.\n\n Try reduceing your values for characteristics to fix \

the problem.'**);**

**}**

***if*** **(**queenFoodCost **-** specieCost **<** 0**)** **{**

window**.**alert**(**'Warning #2:\nThe speciesCost (' **+** specieCost**.**toFixed**(**0**)** **+** ') is greater then \

the queen\'s food cost (' **+** queenFoodCost **+** '. This means that queen ants \

will die immediatly when born.\n\n Try reduceing your values for characteristics to fix \

the problem.'**);**

**}**

***if*** **(**soldierFoodCost **-** specieCost **<** 0**)** **{**

window**.**alert**(**'Warning #3:\nThe speciesCost (' **+** specieCost**.**toFixed**(**0**)** **+** ') is greater then \

the soldier\'s food cost (' **+** soldierFoodCost **+** '. This means that soldier ants \

will die immediatly when born.\n\n Try reduceing your values for characteristics to fix \

the problem.'**);**

**}**

***var*** queenStepsMax **=** parseInt**(**CHARS**.**queenStepsMax**.**value**);**

***var*** queenStepsMin **=** parseInt**(**CHARS**.**queenStepsMin**.**value**);**

***if*** **(**element**.**name **===** 'queenStepsMin' **||** element**.**name **===** 'queenStepsMax'**)** **{**

***if*** **(**queenStepsMin **>** queenStepsMax**)** **{**

window**.**alert**(**'Error #1:\nThe minimum number of Queen steps (' **+** queenStepsMin **+** ') is greater \

then the maximum (' **+** queenStepsMax **+** '). \n\nPlease reduce the minimum number or increase the\

maximum to fix the problem.'**);**

**}**

**}**

// Update the characteristics value in settings as well as the input

getElement**(**characteristic**.**id **+** '-value'**).**innerHTML **=** value**;**

setValue**(**characteristic**.**id**,** value**);**

// Update the species cost

workerCostTotal **=** **(**workerFoodCost **-** specieCost**).**toFixed**(**0**);**

soldierCostTotal **=** **(**soldierFoodCost **-** specieCost**).**toFixed**(**0**);**

queenCostTotal **=** **(**queenFoodCost **-** specieCost**).**toFixed**(**0**);**

getElement**(**'ant-health'**).**innerHTML **=** 'Worker: ' **+** workerCostTotal **+**

'</br>Soldier: ' **+** soldierCostTotal **+**

'</br>Queen: ' **+** queenCostTotal

**}**

/\*\*

\* Updates the value of all characteristics to their default values

\*/

***function*** updateDefaultValues**()** **{**

***for*** **(*var*** prop ***in*** CHARS**)** **{**

***var*** characteristic **=** CHARS**[**prop**];**

updateValue**(**getElement**(**characteristic**.**id**),** characteristic**.**defaultValue**);**

**}**

**}**

/\*\*

\* Updates the value of all characteristics a random value

\*/

***function*** updateRandomValues**()** **{**

***for*** **(*var*** prop ***in*** CHARS**)** **{**

***var*** characteristic **=** CHARS**[**prop**];**

***if*** **(!**characteristic**.**editable**)**

***continue*;**

// Depending on type of value, use either randInt or randFloat

***if*** **(**characteristic**.**type **===** VALUE\_TYPE**.**floatValue**)**

***var*** value **=** randFloat**(**characteristic**);**

***else*** ***if*** **(**characteristic**.**type **===** VALUE\_TYPE**.**integerValue**)**

***var*** value **=** randInt**(**characteristic**);**

updateValue**(**getElement**(**characteristic**.**id**),** value**);**

**}**

**}**

/\*\*

\* Create a new HTML element

\* @param {string} tag - A HTML tag name e.g. 'div'

\* @param [{type : string, value : \*}] attributes - A list of attributes to

\* add to the new element

\* @return {HTML element}

\*/

***function*** newElement**(**tag**,** attributes**)** **{**

***var*** element **=** document**.**createElement**(**tag**);** // create element object

***for*** **(*var*** i **=** 0**;** i **<** attributes**.**length**;** i**++)** // give element attributes

element**.**setAttribute**(**attributes**[**i**].**type**,** attributes**[**i**].**value**);**

***return*** element**;**

**}**

/\*\*

\* Create an input element of type [range, button, colour]

\* @param {\*} characteristic - A single characteristic from CHARS variable

\* @param {\*} prop - The property of CHARS which the characteristic refers to

\* @return {HTML element}

\*/

***function*** createInputType**(**characteristic**,** prop**)** **{**

// Create the intial input tag

***var*** input **=** newElement**(**'input'**,** **[{**

type**:** 'class'**,**

value**:** 'config'

**},** **{**

type**:** 'type'**,**

value**:** 'range'

**},** **{**

type**:** 'id'**,**

value**:** characteristic**.**id

**},** **{**

type**:** 'name'**,**

value**:** prop

**},** **{**

type**:** 'onchange'**,**

value**:** 'updateValue(this, this.value);'

**},** **]);**

***if*** **(!**characteristic**.**editable**)** input**.**setAttribute**(**'disabled'**,** 'disabled'**);**

input**.**value **=** characteristic**.**defaultValue**;**

// Depending on the type add extra attributes

***if*** **(**characteristic**.**inputType **===** INPUT\_TYPE**.**slider**)** **{**

input**.**setAttribute**(**'min'**,** characteristic**.**min**);**

input**.**setAttribute**(**'max'**,** characteristic**.**max**);**

input**.**setAttribute**(**'step'**,** characteristic**.**step**);**

input**.**setAttribute**(**'type'**,** 'range'**);**

**}**

***return*** input**;**

**}**

/\*\*

\* Create a row containing a label, input and value elements used for

\* creating dynamic inputs for characteristics

\* @param {\*} characteristic - A single characteristic from CHARS variable

\* @param {string} prop - The property of CHARS which the characteristic refers to

\* @return {HTML element}

\*/

***function*** createInput**(**characteristic**,** prop**)** **{**

// Row

***var*** row **=** newElement**(**'tr'**,** **[{**

type**:** 'class'**,**

value**:** 'config'

**},** **{**

type**:** 'title'**,**

value**:** characteristic**.**desc

**}]);**

// Label element

***var*** label **=** newElement**(**'td'**,** **[{**

type**:** 'class'**,**

value**:** 'config'

**}]);**

label**.**innerHTML **=** characteristic**.**neatName**;**

// Input and container elements

***var*** inputContainer **=** newElement**(**'td'**,** **[{**

type**:** 'class'**,**

value**:** 'config'

**},** **{**

type**:** 'id'**,**

value**:** 'input-container'

**}]);**

***var*** input **=** createInputType**(**characteristic**,** prop**);**

// Value element

***var*** value **=** newElement**(**'td'**,** **[{**

type**:** 'class'**,**

value**:** 'config'

**},** **{**

type**:** 'id'**,**

value**:** characteristic**.**id **+** '-value'

**}]);** // Append '-value' to the id of the elements which show the characteristics value

***if*** **(**characteristic**.**type **===** VALUE\_TYPE**.**floatValue**)**

value**.**innerHTML **=** characteristic**.**value**.**toFixed**(**NUMBER\_OF\_FIXED\_PLACES**);**

***else***

value**.**innerHTML **=** characteristic**.**value**;**

// Add all the individual elements to the row

inputContainer**.**appendChild**(**input**);**

row**.**appendChild**(**label**);**

row**.**appendChild**(**inputContainer**);**

row**.**appendChild**(**value**);**

***return*** row**;**

**}**

/\*\*

\* Creates inputs for all characteristics used for creating dynamic

\* inputs for characteristics

\* And appends them to a table in the config panel

\*/

***function*** createCharacteristicInputs**()** **{**

***var*** configPanel **=** getElement**(**'config'**);**

***var*** table **=** newElement**(**'table'**,** **{**

type**:** 'class'**,**

value**:** 'config'

**});**

// For each characteristic in CHARS, create its input row and add to the

// configuration table.

***for*** **(*var*** prop ***in*** CHARS**)** **{**

***var*** inputRow **=** createInput**(**CHARS**[**prop**],** prop**);**

table**.**appendChild**(**inputRow**);**

**}**

configPanel**.**appendChild**(**table**);**

**}**

/\*\*

\* Updates the selected species with the new values selected in the config panel

\*/

***function*** updateUserSpecies**()** **{**

// Return the button to its original colour showing no pending updates

// need to be pushed to the species

getElement**(**'button-update'**).**style**.**color **=** BUTTON\_NO\_UPDATE\_COLOUR**;**

***var*** specieCost **=** 0**;**

// For each characteristic in CHARS set its input rows, value field to its

// value

***for*** **(*var*** prop ***in*** CHARS**)** **{**

***var*** characteristic **=** CHARS**[**prop**];**

// Parse the input to make sure its in the correct format

***if*** **(**characteristic**.**type **===** VALUE\_TYPE**.**floatValue**)**

***var*** value **=** parseFloat**(**characteristic**.**value**);**

***else*** ***if*** **(**characteristic**.**type **===** VALUE\_TYPE**.**integerValue**)**

***var*** value **=** parseInt**(**characteristic**.**value**);**

// Update the selected species characteristics to this new value

SELECTED\_SPECIES**.**chars**[**prop**]** **=** value**;**

specieCost **+=** CHARS**[**prop**].**healthModifier **\*** value**;**

**}**

// Recalculate the cost for workers, soldiers and queens.

workerCostTotal **=** **(**CHARS**.**reproductionWorkerFoodCost**.**value **\*** FOOD\_HEALTH\_RATIO **-** specieCost**).**toFixed**(**0**);**

soldierCostTotal **=** **(**CHARS**.**reproductionSoldierFoodCost**.**value **\*** FOOD\_HEALTH\_RATIO **-** specieCost**).**toFixed**(**0**);**

queenCostTotal **=** **(**CHARS**.**reproductionQueenFoodCost**.**value **\*** FOOD\_HEALTH\_RATIO **-** specieCost**).**toFixed**(**0**);**

getElement**(**'ant-health'**).**innerHTML **=** 'Worker: ' **+** workerCostTotal **+**

'</br>Soldier: ' **+** soldierCostTotal **+**

'</br>Queen: ' **+** queenCostTotal

**}**

/\*\*

\* Create a row containing a label and data elements used for displaying

\* information about species

\* @param {string} className - The name of the class for the species

\* @param {string} id - The ID which will be used to access and update the species

\* @param {string} labelValue - The text which will be used as a label for the data

\* @param {string} dataValue - The data which will be displayed

\* @return {HTML element}

\*/

***function*** createDataRow**(**className**,** id**,** labelValue**,** dataValue**)** **{**

***var*** row **=** newElement**(**'tr'**,** **[{**

type**:** 'class'**,**

value**:** className

**}]);**

***var*** label **=** newElement**(**'td'**,** **[{**

type**:** 'class'**,**

value**:** className

**}]);**

label**.**innerHTML **=** labelValue**;**

***var*** data **=** newElement**(**'td'**,** **[{**

type**:** 'class'**,**

value**:** className

**},** **{**

type**:** 'id'**,**

value**:** id **+** '-data'

**}]);** // Always append '-data' so can be eddied later

data**.**innerHTML **=** dataValue**;**

row**.**appendChild**(**label**);**

row**.**appendChild**(**data**);**

***return*** row**;**

**}**

/\*\*

\* Crate all elements to display information about a species

\* @param {species object} species - The species whose data you want to create

\* a data display for

\*/

***function*** createSpeciesData**(**species**)** **{**

***var*** id **=** species**.**id**;**

***var*** className **=** 'species' **+** ' ' **+** id**;**

// Create a new table element which will contain all other elements

***var*** table **=** newElement**(**'table'**,** **[{**

type**:** 'class'**,**

value**:** className

**}]);**

// Create a new row for the title and visibility button

***var*** titleRow **=** newElement**(**'tr'**,** **[{**

type**:** 'class'**,**

value**:** className

**},** **{**

type**:** 'id'**,**

value**:** id **+** '-label-row'

**}]);** // Given a specific ID so does not turn invisible when visibility is toggled

titleRow**.**style**.**color **=** species**.**colour**.**nest**;**

***var*** title **=** newElement**(**'td'**,** **[{**

type**:** 'class'**,**

value**:** className

**},** **{**

type**:** 'onclick'**,**

value**:** 'select(this)'

**}]);**

title**.**innerHTML **=** 'Species: ' **+** id**;**

***var*** toggleVisibility **=** newElement**(**'td'**,** **[{**

type**:** 'class'**,**

value**:** className **+** ' toggleVisibility'

**},** **{**

type**:** 'onclick'**,**

value**:** 'toggleClassVisibility(this)'

**}]);**

toggleVisibility**.**innerHTML **=** '-'**;** // Default not expanded

// Create a data row for colour

***var*** colourRow **=** createDataRow**(**className**,** id **+** '-colour'**,** 'Colour'**,** ''**);**

colourRow**.**childNodes**[**1**].**style**.**backgroundColor **=** species**.**colour**.**nest**;**

// Create a data row for number of ants

***var*** antNumRow **=** createDataRow**(**className**,** id **+** '-antNum'**,** 'Number of ants'**,**

species**.**ants**.**length**);**

// Create a data row for number of nests

***var*** nestNumRow **=** createDataRow**(**className**,** id **+** '-nestNum'**,** 'Number of Nests'**,**

species**.**nests**.**length**);**

// Create a data row for amount of food

***var*** foodAmount **=** 0**;**

***for*** **(*var*** i **=** 0**;** i **<** species**.**nests**.**length**;** i**++)**

foodAmount **+=** species**.**nests**[**i**].**health**;**

***var*** foodAmountRow **=** createDataRow**(**className**,** id **+** '-foodAmount'**,** 'Amount of food'**,**

foodAmount**);**

// Append all of the rows to the table

titleRow**.**appendChild**(**title**);**

titleRow**.**appendChild**(**toggleVisibility**);**

table**.**appendChild**(**titleRow**);**

table**.**appendChild**(**colourRow**);**

table**.**appendChild**(**antNumRow**);**

table**.**appendChild**(**nestNumRow**);**

table**.**appendChild**(**foodAmountRow**);**

// Append the table to the data panel

getElement**(**'data'**).**appendChild**(**table**);**

// Minimize the species

***var*** button **=** document**.**getElementsByClassName**(**className**)[**0**].**getElementsByClassName**(**'toggleVisibility'**)[**0**];**

toggleClassVisibility**(**button**);**

**}**

/\*\*

\* Updates data for all of the species currently in the simulation

\*/

***function*** updateSpeciesData**()** **{**

***for*** **(*var*** i **=** 0**;** i **<** SPECIES\_LIST**.**length**;** i**++)** **{**

***var*** species **=** SPECIES\_LIST**[**i**];**

***var*** id **=** species**.**id**;**

// Update the species colour

***var*** colourDataElement **=** getElement**(**id **+** '-colour-data'**);**

colourDataElement**.**style**.**backgroundColour **=** species**.**colour**.**nest**;**

// Update the number of Ants

***var*** antNumDataElement **=** getElement**(**id **+** '-antNum-data'**);**

antNumDataElement**.**innerHTML **=** species**.**ants**.**length**;**

// Update the number of Nests

***var*** nestNumDataElement **=** getElement**(**id **+** '-nestNum-data'**);**

nestNumDataElement**.**innerHTML **=** species**.**nests**.**length**;**

// Update the amount of food however every AVERAGE\_FOOD\_SAMPLE\_RATE ticks

***if*** **(**CURRENT\_TICK **%** AVERAGE\_FOOD\_SAMPLE\_RATE **===** 0**)** **{**

***var*** foodAmountDataElement **=** getElement**(**id **+** '-foodAmount-data'**);**

***var*** foodAmount **=** 0**;**

***for*** **(*var*** k **=** 0**;** k **<** species**.**nests**.**length**;** k**++)**

foodAmount **+=** species**.**nests**[**k**].**health**;**

console**.**log**(**foodAmount**)**

foodAmountDataElement**.**innerHTML **=** foodAmount**.**toFixed**(**NUMBER\_OF\_FIXED\_PLACES**);**

**}**

**}**

**}**

/\*\*

\* Removes a species data i.e. when it has died out

\* @param {number} id - The id of the species to remove

\*/

***function*** removeSpeciesData**(**id**)** **{**

document**.**getElementsByClassName**(**id**)[**0**].**innerHTML **=** ''**;**

**}**

/\*\*

\* Checks whether the species data is currently maximised or minimised and returns

\* whether or not the species data should be hidden or not assuming the button has

\* been clicked

\* @param {HTML element} button - A reference to the toggle visibility button

\* @return {string} - The value of the display styling

\*/

***function*** nextVisibility**(**button**)** **{**

***if*** **(**button**.**innerHTML **===** '+'**)** ***return*** 'table-row'**;** // Show the element

***else*** ***return*** 'none'**;** // Hide the element

**}**

/\*\*

\* Toggles the text in the visibility button

\* @param {HTML element} button - A reference to the toggle visibility button

\*/

***function*** toggleVisibilityButton**(**button**)** **{**

***if*** **(**button**.**innerHTML **===** '-'**)** button**.**innerHTML **=** '+'**;**

***else*** button**.**innerHTML **=** '-'**;**

**}**

/\*\*

\* Maximises and Minimises the data for a particular species

\* @param {HTML element} button - A reference to the toggle visibility button

\*/

***function*** toggleClassVisibility**(**button**)** **{**

***var*** id **=** button**.**className**.**split**(**' '**)[**1**];** // The second class name is the species id

// Get all other elements with the same class name i.e. all elements relating to that species data

***var*** listOfClassElements **=** document**.**getElementsByClassName**(**id**)[**0**].**getElementsByTagName**(**'TR'**);**

// For each element in the list

***for*** **(*var*** i **=** 0**;** i **<** listOfClassElements**.**length**;** i**++)** **{**

// Check if its a row and also check if its not the row containing the visibility

// button (so as not to hide the button from being used afterwards)

***if*** **(**listOfClassElements**[**i**].**id **!==** id **+** '-label-row'**)** **{**

listOfClassElements**[**i**].**style**.**display **=** nextVisibility**(**button**);** // Then hide/show the element

**}**

**}**

// Finally toggle the button to show the next symbol i.e. + or -

toggleVisibilityButton**(**button**);**

**}**

/\*\*

\* Selects a new species allowing the user to alter that species characteristics

\* @param {HTML element} title - A reference to the title element of the species data

\*/

***function*** select**(**title**)** **{**

// De colour previous selected table

document**.**getElementsByClassName**(**SELECTED\_SPECIES**.**id**)[**0**].**style**.**backgroundColor **=** UNSELECTED\_COLOUR**;**

// Update new selected table

***var*** id **=** title**.**className**.**split**(**' '**)[**1**];** // The second class name is the species id

document**.**getElementsByClassName**(**id**)[**0**].**style**.**backgroundColor **=** SELECTED\_COLOUR**;**

// Find the species which the id refers to and set the selected species to it

***for*** **(*var*** s ***in*** SPECIES\_LIST**)** **{**

***var*** species **=** SPECIES\_LIST**[**s**];**

***if*** **(**species**.**id **===** parseInt**(**id**))** **{**

SELECTED\_SPECIES **=** species**;**

**}**

**}**

// Update the characteristics in the config panel to the characteristics of the selected species

***for*** **(*var*** prop ***in*** CHARS**)** **{**

***var*** characteristic **=** CHARS**[**prop**];**

updateValue**(**getElement**(**characteristic**.**id**),** SELECTED\_SPECIES**.**chars**[**prop**]);**

**}**

***if*** **(**SELECTED\_SPECIES**.**nests**.**length **>** 0**)** **{**

CANVAS\_OFFSET**.**x **=** **-**SELECTED\_SPECIES**.**nests**[**0**].**coord**.**x **\*** CELL\_SIZE**.**width **+** CANVAS**.**width **/** 2**;**

CANVAS\_OFFSET**.**y **=** **-**SELECTED\_SPECIES**.**nests**[**0**].**coord**.**y **\*** CELL\_SIZE**.**width **+** CANVAS**.**width **/** 2**;**

**}**

**}**

## core/canvas.js

/\*\*

\* Resizes a HTML element to a specific size

\* @param {HTML element} element - The element to be resized

\* @param {width : number, height : number} size - The new size of the element

\*/

***function*** resizeElement**(**element**,** size**)** **{**

element**.**width **=** size**.**width**;**

element**.**height **=** size**.**height**;**

**}**

/\*\*

\* Draws a rectangle onto a canvas context

\* @param {canvas context 2d} ctx - The context which the rectangle will be drawn onto

\* @param {x : number, y : number} coord - The coordinate of the top left corner

\* @param {width : number, height : number} size - The size of the rectangle

\* @param {string} fillColour - The colour of the rectangle

\* @param {string} strokeColour - The stroke colour of the rectangle

\* (defaults to '#000000')

\* @param {number} lineWidth - The width of the border (defaults to 0)

\*/

***function*** drawRect**(**ctx**,** coord**,** size**,** fillColour**,** strokeColour**,** lineWidth**)** **{**

***if*** **(*typeof*** strokeColour **===** 'undefined'**)** strokeColour **=** '#000000'**;**

***if*** **(*typeof*** lineWidth **===** 'undefined'**)** lineWidth **=** 0**;**

ctx**.**fillStyle **=** fillColour**;**

ctx**.**fillRect**(**coord**.**x**,** coord**.**y**,** size**.**width**,** size**.**height**);**

// Only boarder if width is > 0

***if*** **(**lineWidth **>** 0**)** **{**

ctx**.**strokeStyle **=** strokeColour**;**

ctx**.**lineWidth **=** lineWidth**;**

ctx**.**stroke**();**

**}**

**}**

/\*\*

\* Draws a line onto a canvas context

\* @param {canvas context 2d} ctx - The context which the line will be drawn onto

\* @param {x : number, y : number} coord1 - The coordinate of the starting point

\* @param {x : number, y : number} coord2 - The coordinate of the ending point

\* @param {string} strokeColour - The stroke colour of the line

\* @param {number} lineWidth - The width of the stroke

\*/

***function*** drawLine**(**ctx**,** coord1**,** coord2**,** strokeColour**,** lineWidth**)** **{**

// Drawing straight lines only

// Only boarder if width is > 0

***if*** **(**lineWidth **>** 0**)** **{**

ctx**.**strokeStyle **=** strokeColour**;**

ctx**.**lineWidth **=** lineWidth**;**

ctx**.**beginPath**();**

ctx**.**moveTo**(**coord1**.**x**,** coord1**.**y**);**

ctx**.**lineTo**(**coord2**.**x**,** coord2**.**y**);**

ctx**.**closePath**();**

ctx**.**stroke**();**

**}**

**}**

/\*\*

\* Draws an arc onto a canvas context

\* @param {canvas context 2d} ctx - The context which the arc will be drawn onto

\* @param {x : number, y : number} coord - The coordinate of the centre of the arc

\* @param {number} radius - The radius of the arc

\* @param {number} startAngle - The angle from the horizontal to start the arc at

\* @param {number} endAngle - The angle from the horizontal to stop the arc at

\* @param {string} strokeColour - The stroke colour of the arc

\* @param {number} lineWidth - The width of the stroke

\* @param {string} fillColour - The colour of the arc

\*/

***function*** drawArc**(**ctx**,** coord**,** radius**,** startAngle**,** endAngle**,** strokeColour**,**

lineWidth**,** fillColour**)** **{**

***if*** **(**lineWidth **>** 0**)** **{**

ctx**.**beginPath**();**

ctx**.**arc**(**coord**.**x**,** coord**.**y**,** radius**,** startAngle**,** endAngle**,** ***false*);**

ctx**.**closePath**();**

***if*** **(*typeof*** strokeColour **!==** 'unefined'**)** **{**

ctx**.**strokeStyle **=** strokeColour**;**

ctx**.**lineWidth **=** lineWidth**;**

ctx**.**stroke**();**

**}**

***if*** **(*typeof*** fillColour **!==** 'unefined'**)** **{**

ctx**.**fillStyle **=** fillColour**;**

ctx**.**fill**();**

**}**

**}**

**}**

/\*\*

\* Draws a circle

\* @param {canvas context 2d} ctx - The context which the arc will be drawn onto

\* @param {x : number, y : number} coord - The coordinate of the centre of the arc

\* @param {number} radius - The radius of the arc

\* @param {string} fillColour - The colour of the arc

\*/

***function*** drawCircle**(**ctx**,** coord**,** radius**,** fillColour**)** **{**

ctx**.**beginPath**();**

ctx**.**arc**(**coord**.**x**,** coord**.**y**,** radius**,** 0**,** 2 **\*** Math**.**PI**,** ***false*);**

ctx**.**closePath**();**

ctx**.**fillStyle **=** fillColour**;**

ctx**.**fill**();**

**}**

/\*\*

\* Draws a rectangle over the entire canvas effectively wiping it to a single colour

\* @param {canvas context 2d} ctx - The context which the rectangle will be drawn onto

\*/

***function*** clearCanvas**(**ctx**)** **{**

drawRect**(**ctx**,** **{**

x**:** 0**,**

y**:** 0

**},** CANVAS**,** OUT\_OF\_BOUNDS\_COLOUR**);**

**}**

## core/utilities.js

/\*\*

\* Returns a random integer within a specific range

\* @param {min : integer, max : integer} range - The range inclusive

\* @return {integer}

\*/

***function*** randInt**(**range**)** **{**

***return*** Math**.**floor**(**Math**.**random**()** **\*** **(**range**.**max **-** range**.**min **+** 1**)** **+** range**.**min**);**

**}**

/\*\*

\* Returns a random float within a specific range

\* @param {min : number, max : number} range - The range inclusive

\* @return {number}

\*/

***function*** randFloat**(**range**)** **{**

***if*** **(**Math**.**random**()** **<** 0.5**)**

***var*** result **=** **(**1 **-** Math**.**random**())** **\*** **(**range**.**max **-** range**.**min**)** **+** range**.**min**;**

***else***

***var*** result **=** **(**Math**.**random**()** **\*** **(**range**.**max **-** range**.**min**)** **+** range**.**min**);**

***return*** result**;**

**}**

/\*\*

\* Returns a random angle between 0 and 2PI radians

\* @return {number}

\*/

***function*** randDir**()** **{**

***return*** randFloat**({**

min**:** 0**,**

max**:** Math**.**PI **\*** 2

**});**

**}**

/\*\*

\* Returns a random hex colour

\* @return {string}

\*/

***function*** randColour**()** **{**

***return*** '#' **+** Math**.**floor**(**Math**.**random**()** **\*** 16777215**).**toString**(**16**);**

**}**

/\*\*

\* Returns an angle in the range 0 to 2PI i.e. if dir = 4PI returns 2PI

\* @param {number} dir - The direction

\* @return {number}

\*/

***function*** validateDirection**(**dir**)** **{**

***return*** dir **-** Math**.**PI **\*** 2 **\*** Math**.**floor**(**dir **/** **(**Math**.**PI **\*** 2**));**

**}**

/\*\*

\* Picks a random property of a object literal

\* @param {object} obj - the object literal

\* @return {object}

\*/

***function*** randProperty**(**obj**)** **{**

***var*** result**;**

***var*** count **=** 0**;**

***for*** **(*var*** prop ***in*** obj**)**

***if*** **(**Math**.**random**()** **<** 1 **/** **++**count**)**

result **=** prop**;**

***return*** result**;**

**}**

/\*\*

\* Returns the shortest distance between two coordinates

\* @param {x : number, y : number} coord1, coord2 - The coordinates

\* @return {number}

\*/

***function*** distance**(**coord1**,** coord2**)** **{**

// picks the minimum distance either wraping around the map or directly

***var*** x **=** Math**.**min**(**Math**.**abs**(**coord1**.**x **-** coord2**.**x**),** GRID\_SIZE**.**width **-** Math**.**abs**(**coord1**.**x **-** coord2**.**x**));**

***var*** y **=** Math**.**min**(**Math**.**abs**(**coord1**.**y **-** coord2**.**y**),** GRID\_SIZE**.**height **-** Math**.**abs**(**coord1**.**y **-** coord2**.**y**));**

***return*** Math**.**sqrt**(**x **\*** x **+** y **\*** y**);**

**}**

/\*\*

\* Scale a coordinate to its location in pixels

\* @param {x : number, y : number} coord - The coordinate to scale

\* @return {x : number, y : number} - The scaled coordinate

\*/

***function*** scaleCoord**(**coord**)** **{**

***return*** **{**

x**:** coord**.**x **\*** CELL\_SIZE**.**width **+** CANVAS\_OFFSET**.**x**,**

y**:** coord**.**y **\*** CELL\_SIZE**.**height **+** CANVAS\_OFFSET**.**y

**};**

**}**

/\*\*

\* Converts a coordinate to a map index, it can be used with coordinates which

\* don't map exactly to a single index and will work out which cell the coordinate

\* lies in mostly.

\* @param {x : number, y : number} coord - The coordinate which will be converted

\* @return {integer} - The map index which the coordinate converts to

\*/

***function*** coordToIndex**(**coord**)** **{**

***var*** cellCoord **=** boundary**({**

x**:** Math**.**round**(**coord**.**x**),**

y**:** Math**.**round**(**coord**.**y**)**

**},** MAP\_BOUNDARY**);**

***return*** cellCoord**.**x **+** cellCoord**.**y **\*** GRID\_SIZE**.**width**;**

**}**

/\*\*

\* Converts a map index to a coordinate

\* @param {integer} index - The map index

\* @return {x : number, y : number} - The coordinate which the index converts to

\*/

***function*** indexToCoord**(**index**)** **{**

// Translates linear position in map to (x, y) coordinates on grid

***var*** x **=** index **-** Math**.**floor**(**index **/** GRID\_SIZE**.**width**)** **\*** GRID\_SIZE**.**height**;**

***var*** y **=** Math**.**floor**(**index **/** GRID\_SIZE**.**width**);**

***return*** **{**

x**:** x**,**

y**:** y

**};**

**}**

/\*\*

\* Smiler to coordToIndex however returns the \*neat\* coordinate i.e. a coordinate

\* which maps exactly to a single map index by finding the cell which the

\* coordinate lies in mostly

\* @param {x : number, y : number} coord - The coordinate

\* @return {x : number, y : number} - The coordinate which maps exactly to a single map index

\*/

***function*** getCellCoord**(**coord**)** **{**

***return*** boundary**({**

x**:** Math**.**floor**(**coord**.**x**),**

y**:** Math**.**floor**(**coord**.**y**)**

**},** MAP\_BOUNDARY**);**

**}**

/\*\*

\* Returns the \*warped\* coordinate of a coordinate if it exceeds the map boundary.

\* As the map wraps around if an ant goes off one side it will appear on the other.

\* @param {x : number, y : number} coord - The coordinate to test

\* @param {x : {min : number, max : number}, y : {min : number, max : number}} bounds

\* - The boundary which is tested against

\* @return {x : number, y : number} - The warped coordinate

\*/

***function*** boundary**(**coord**,** bounds**)** **{**

// Check x bounds

***if*** **(**coord**.**x **<** bounds**.**x**.**min**)**

coord**.**x **=** bounds**.**x**.**max **-** Math**.**abs**(**coord**.**x**);** // x may be negative if x < 0

***else*** ***if*** **(**coord**.**x **>=** bounds**.**x**.**max**)**

coord**.**x **=** coord**.**x **-** bounds**.**x**.**max**;** // x cannot be negative unless boundary is

// Check y bounds

***if*** **(**coord**.**y **<** bounds**.**y**.**min**)**

coord**.**y **=** bounds**.**y**.**max **-** Math**.**abs**(**coord**.**y**);**

***else*** ***if*** **(**coord**.**y **>=** bounds**.**y**.**max**)**

coord**.**y **=** coord**.**y **-** bounds**.**y**.**max**;**

***return*** coord**;**

**}**

/\*\*

\* Returns an array of cells which lie a certain distance around a specific point

\* @param {x : number, y : number} coord - The coordinate to get the block around

\* @param {width : integer, height : integer} size - The size of the block i.e.

\* if width = 2, takes 2 block to the left, and two blocks to the

\* right of the coordinate

\* @return {[{x : number, y : number}]} - An array of coordinates which lie

\* around the coordinate

\*/

***function*** getBlock**(**coord**,** size**)** **{**

block **=** **[];**

***for*** **(*var*** y **=** coord**.**y **-** size**.**height**;** y **<=** coord**.**y **+** size**.**height**;** y**++)** **{**

***for*** **(*var*** x **=** coord**.**x **-** size**.**width**;** x **<=** coord**.**x **+** size**.**width**;** x**++)** **{**

block**.**push**(**getCellCoord**({**

x**:** x**,**

y**:** y

**}));**

**}**

**}**

***return*** block**;**

**}**

/\*\*

\* Returns an array of cells which lie in the sector of a circle of a

\* particular radius around a specific point

\* @param {x : number, y : number} coord - The coordinate to get the block around

\* @param {integer} radius - The radius of the circle which a sector is being taken from

\* @param {number} direction - The direction the ant is facing

\* @param {number} angle - The angle of the sector

\* @return {[{x : number, y : number}]} - An array of coordinates which lie in the sector

\*/

***function*** getSector**(**coord**,** radius**,** direction**,** angle**)** **{**

***var*** block **=** **[];**

***for*** **(*var*** y **=** coord**.**y **-** radius**;** y **<=** coord**.**y **+** radius**;** y**++)** **{**

***for*** **(*var*** x **=** coord**.**x **-** radius**;** x **<=** coord**.**x **+** radius**;** x**++)** **{**

***var*** searchCoord **=** **{**

x**:** x**,**

y**:** y

**};**

***var*** validDir **=** validateDirection**(**direction**);**

***var*** theta **=** validateDirection**(**angleTo**(**coord**,** searchCoord**));**

***var*** dist **=** distance**(**coord**,** searchCoord**);**

***var*** minSector **=** validateDirection**(**direction **-** angle **/** 2**);**

***var*** maxSector **=** validateDirection**(**direction **+** angle **/** 2**);**

// Determine if the coordinate lies in the sector

***if*** **(**theta **>=** minSector **&&** theta **<=** maxSector **&&** dist **<=** radius**)** **{**

block**.**push**(**getCellCoord**(**searchCoord**));**

**}** ***else*** ***if*** **((**validDir **<=** angle **/** 2 **||** validDir **>=** Math**.**PI **\*** 2 **-** angle **/** 2**)** **&&**

**(**theta **<=** maxSector **||** theta **>=** minSector**)** **&&** dist **<=** radius**)** **{**

block**.**push**(**getCellCoord**(**searchCoord**));**

**}**

**}**

**}**

***return*** block**;**

**}**

/\*\*

\* Returns the reverse direction i.e. + PI radians

\* @param {number} angle - The angle in radians

\* @return {number}

\*/

***function*** turnAround**(**angle**)** **{**

***return*** angle **+** Math**.**PI**;**

**}**

/\*\*

\* Returns the direction/angle of shortest path to get from the coord to the target

\* @param {x : number, y : number} coord, target - The coordinates

\* @return {number} - The angel from the vertical axis clockwise in radians

\*/

***function*** angleTo**(**coord**,** target**)** **{**

// Find the minimum distance and go in that direction i.e. either

// directly to target or warping around map

***if*** **(**GRID\_SIZE**.**width **-** Math**.**abs**(**target**.**x **-** coord**.**x**)** **>** Math**.**abs**(**target**.**x **-** coord**.**x**))** **{**

***var*** dx **=** target**.**x **-** coord**.**x**;**

**}** ***else*** **{**

***var*** dx **=** GRID\_SIZE**.**width **-** **(**target**.**x **-** coord**.**x**);**

**}**

***if*** **(**GRID\_SIZE**.**height **-** Math**.**abs**(**target**.**y **-** coord**.**y**)** **>** Math**.**abs**(**target**.**y **-** coord**.**y**))** **{**

***var*** dy **=** target**.**y **-** coord**.**y**;**

**}** ***else*** **{**

***var*** dy **=** GRID\_SIZE**.**height **-** **(**target**.**y **-** coord**.**y**);**

**}**

***return*** Math**.**atan2**(**dy**,** dx**)** **+** Math**.**PI **/** 2**;** // atan2 find the angle from the horizontal

// however ants use angle from the vertical

**}**

/\*\*

\* Creates a new ant

\* @param {Species object} species - The species of the new ant

\* @param {x : number, y : number} coord - The coordinate of the new ant

\* @param {Nest object} nest - The new ants home nest (the nest which it delivers food to)

\* @param {number} startingHealth - The new ants health

\* @param {integer} type - The type of ant to create i.e. ANT\_TYPE.worker

\*/

***function*** createAnt**(**species**,** coord**,** nest**,** startingHealth**,** type**)** **{**

***switch*** **(**type**)** **{**

***case*** ANT\_TYPE**.**worker**:**

***var*** ant **=** ***new*** Worker**(**genID**(),** coord**);**

ant**.**colour **=** species**.**colour**.**worker**;**

***break*;**

***case*** ANT\_TYPE**.**queen**:**

***var*** ant **=** ***new*** Queen**(**genID**(),** coord**);**

ant**.**colour **=** species**.**colour**.**queen**;**

***break*;**

***case*** ANT\_TYPE**.**soldier**:**

***var*** ant **=** ***new*** Soldier**(**genID**(),** coord**);**

ant**.**colour **=** species**.**colour**.**soldier**;**

***break*;**

**}**

// Only possible to mutate queens (due to the nature of ant reproduction)

***if*** **(**type **===** ANT\_TYPE**.**queen**)** **{**

ant**.**species **=** species**.**mutate**();**

**}** ***else*** **{**

ant**.**species **=** species**;**

**}**

ant**.**nest **=** nest**;**

ant**.**species**.**ants**.**push**(**ant**);**

ant**.**health **=** startingHealth**;**

ANTS\_LIST**.**push**(**ant**);**

ant**.**addToMap**();**

**}**

/\*\*

\* Generates a unique id. Requires CURRENT\_ID variable to keep track of current id

\* @return {integer} - A unique ID

\*/

***function*** genID**()** **{**

CURRENT\_ID **+=** 1**;**

***return*** CURRENT\_ID**;**

**}**

/\*\*

\* Clones a object needed as JavaScript passes everything by reference

\* @param {object} obj - The object which will be cloned

\* @return {object} - A copy of object

\*

\* Credit:

\* http://stackoverflow.com/questions/7574054/javascript-how-to-pass-object-by-value

\*/

***function*** clone**(**obj**)** **{**

***if*** **(**obj **===** null **||** ***typeof*** **(**obj**)** **!==** 'object'**)** ***return*** obj**;**

***var*** temp **=** ***new*** obj**.**constructor**();**

***for*** **(*var*** key ***in*** obj**)**

temp**[**key**]** **=** clone**(**obj**[**key**]);**

***return*** temp**;**

**}**

/\*\*

\* Returns the html element an ID responds to

\* @param {string} id - HTML id

\* @return {HTML element} - The HTML element

\*/

***function*** getElement**(**id**)** **{**

***return*** document**.**getElementById**(**id**);**

**}**

/\*\*

\* Sets the value of a html element

\* @param {string} id - HTML id

\* @param {string} value

\*/

***function*** setValue**(**id**,** value**)** **{**

document**.**getElementById**(**id**).**value **=** value**;**

**}**

## core/config.js

/\*\*

\* Misc

\*/

***const*** VALUE\_TYPE **=** **{** // Used to signify if a variable is an integer or float

integerValue**:** 0**,**

floatValue**:** 1

**};**

/\*\*

\* Main simulation loop configuration

\*/

***const*** TICK **=** 10**;** // The time between ticks

***var*** CURRENT\_TICK **=** 0**;** // The current tick number

***var*** CURRENT\_ID **=** 0**;** // Used to keep track of the latest unique ID

***var*** ANTS\_LIST **=** **[];** // Holds all ant objects

***var*** SPECIES\_LIST **=** **[];** // Holds all species objects

***var*** RUNNING **=** ***false*;** // Determines whether the simulation is running or not

/\*\*

\* Canvas configuration

\*/

***const*** GRID\_COLOUR **=** '#DDDDDD'**;** // The colour of the maps grid

***const*** GRID\_LINE\_WIDTH **=** 0.2**;** // The width of the grid lines

***const*** BACKGROUND\_COLOUR **=** '#E3DAB3'**;** // The canvas background colour;

***const*** OUT\_OF\_BOUNDS\_COLOUR **=** '#FFFFFF'**;** // Used if panning around off map

***var*** CELL\_SIZE **=** **{** // Size in pixels of a cell

width**:** 6**,**

height**:** 6

**};**

***var*** CANVAS **=** **{** // Used to define characteristics about the simulations canvas

name**:** 'simulation'**,**

width**:** 500**,**

height**:** 500**,**

**};**

/\*\*

\* Input/Output configuration

\*/

***const*** AVERAGE\_FOOD\_SAMPLE\_RATE **=** 40**;** // The number of ticks to wait between

// sampling for averages of food

***const*** NUMBER\_OF\_FIXED\_PLACES **=** 2**;** // The number of numbers after the

// decimal place to display

***var*** SELECTED\_SPECIES**;** // Holds the species whose

// characteristics can be changed

***const*** SELECTED\_COLOUR **=** '#8AC1DE'**;** // The colour of a selected species

***const*** UNSELECTED\_COLOUR **=** '#FFFFFF'**;** // default colour for unselected

***const*** BUTTON\_UPDATE\_COLOUR **=** '#FF0000'**;** // The colour of a button when settings

// have changed and the update needs

// to be pushed to the simulation

***const*** BUTTON\_NO\_UPDATE\_COLOUR **=** '#000000'**;** // The normal colour of a button when

// no settings have been altered

// Char codes for various keys

***const*** LEFT\_ARROW\_KEY **=** 37**;**

***const*** RIGHT\_ARROW\_KEY **=** 39**;**

***const*** UP\_ARROW\_KEY **=** 38**;**

***const*** DOWN\_ARROW\_KEY **=** 40**;**

***const*** PLUS\_KEY **=** 107**;**

***const*** MINUS\_KEY **=** 109**;**

***const*** SPACE\_BAR\_KEY **=** 32**;**

***const*** R\_KEY **=** 82**;**

***const*** S\_KEY **=** 83**;**

***const*** MIN\_ZOOM **=** 1**;** // The minimum zoom amount

***const*** MAX\_ZOOM **=** 14**;** // The maximum zoom amount

***const*** PAN\_AMOUNT **=** 15**;** // The number of pixels to pan by

// each press of an arrow key

***const*** ZOOM\_AMOUNT **=** 0.5**;** // The amount to zoom each key press

***var*** CANVAS\_OFFSET **=** **{** // The amount of pixels the

// canvas is offset

x**:** 0**,**

y**:** 0

**};**

***const*** INPUT\_TYPE **=** **{** // Used for creating custom inputs

// i.e. sliders, buttons

slider**:** 0**,**

button**:** 1**,**

colour**:** 2

**};**

/\*\*

\* Map configuration

\*/

***var*** MAP **=** **[];** // Holds all objects displayed on map

***var*** GRID\_SIZE **=** **{** // Size in number of cells

width**:** 500**,**

height**:** 500

**};**

***var*** MAP\_BOUNDARY **=** **{** // Used to tell if ant is out of bounds

x**:** **{**

min**:** 0**,**

max**:** GRID\_SIZE**.**width

**},**

y**:** **{**

min**:** 0**,**

max**:** GRID\_SIZE**.**height

**}**

**};**

***var*** NUM\_OF\_CELLS **=** GRID\_SIZE**.**width **\*** GRID\_SIZE**.**height**;** // The total number of cells

/\*\*

\* Food configuration

\*/

***const*** FOOD\_COLOUR **=** '#00FF00'**;** // The colour of food

***const*** FOOD\_HEALTH\_RATIO **=** 100**;** // food : health i.e. 1 food worth 50 health

***const*** FOOD\_CHANCE **=** 0.0004**;** // The probability of a food source in a cell

***const*** FOOD\_GROW\_AMOUNT **=** 100**;** // The rate at which food amount increases

***const*** FOOD\_GROW\_RATE **=** 0.01**;** // The probability of each piece of food being

// grown

/\*\*

\* Ant configuration

\*/

***var*** STARTING\_QUEEN\_ANT\_NUMBER **=** 1**;** // The number of queen ants at

// the start of the simulation

***const*** ANT\_FOOD\_DROP\_SPEED **=** 15**;** // The number of ticks for an ant

// to drop a piece of food

***const*** ANT\_FOOD\_TAKE\_SPEED **=** 30**;** // The number of ticks for an ant

// to take a piece of food

***const*** DAMAGE\_MULTIPLIER **=** 100**;** // DAMAGE\_MULTIPLIER \* species.chars.jawStrength

// The amount of health a solider

// ant takes from ants its attacking

***const*** NEST\_GUARD\_RADIUS **=** 100**;** // The number of steps ants take

// once seeing the nest

// i.e. the sentry radius

***const*** SOLDIER\_ANT\_MAX\_TARGET\_DISTANCE **=** 25**;** // The maximum distance between

// an ant and its target before

// losing interest

***const*** TURN\_RATE **=** 0.02**;** // The rate at which ants turn when

// on guarding

***const*** GOAL **=** **{** // Ant goals used to determine ant actions

none**:** **-**1**,** // No current goal, results to default for particular ant

findFood**:** 0**,** // Make ant look for food

getFood**:** 1**,** // Make ant go towards and pick up nearby food

dropFood**:** 2**,** // Make ant look for nest and when can see nest

// deposit food at the nest

pickDirection**:** 3**,** // Used by Queens to pick a direction to travel

gotoNestSite**:** 4**,** // Used by Queens to head in the chosen direction

// towards the nest site

createNest**:** 5**,** // Used by Queens to create a nest

guardNest**:** 6**,** // Used by Soldier ants to guard the nest

guardPheromone**:** 7**,** // Used by Soldier ants to guard pheromone trials

guardFood**:** 8**,** // Used by Soldier ants to guard food

attack**:** 9 // Used by Soldier ants attack other ants

**};**

***const*** NEST\_SIZE **=** **{** // The extra size of a nest in cells e.g. {width : 1, height : 1}

// means go 1 extra cell in both sides and 1 extra cell

// both up and down.

width**:** 1**,**

height**:** 1

**};**

***const*** ANT\_TYPE **=** **{** // Used to show the type of ant

queen**:** 0**,**

worker**:** 1**,**

soldier**:** 2**,**

nest**:** 3**,**

debug**:** 4

**};**

/\*\*

\* Pheromone configuration

\*/

***const*** PHEROMONE\_EVAPERATION\_RATE **=** 0.001**;** // The global amount all pheromone

// concentrations reduce by each tick

***const*** MAX\_PHEROMONE\_CONCENTRATION **=** 1.2**;** // The maximum concentration a

// pheromone can have

/\*\*

\* Species configuration

\*/

***var*** USER\_SPECIES**;** // The species used on first run i.e. the first species

***var*** CHARS **=** **{** // Holds properties of all characteristics species

/\*\*

\* @property {number} min - The minimum values a characteristic can be

\* @property {number} max - The maximum values a characteristic can be

\* @property {integer} type - The type of value a characteristic is

\* i.e. a float or integer

\* @property {string} id - The HTML ID property of the characteristic

\* (used when creating inputs)

\* @property {string} neatName - The name used in a label in the HTML

\* @property {string} desc - Describe what the characteristic does

\* @property {number} step - The minimum change in the characteristics

\* value (used for creating HTML inputs)

\* @property {number} healthModifier - The cost of the characteristic

\* ( = healthModifier \* value)

\* @property {number} defaultValue - The default value

\* @property {number} value - The actual value of the HTML input

\* (differs from the value stored in species

\* as this is formated for displaying)

\* @property {boolean} editable - Determines if a characteristic can be edited

\* @property {integer} inputType- The type of input requied i.e. button

\*

\* Note : see species class for descriptions on effects of individual characteristics

\*/

speed**:** **{**

min**:** 0**,**

max**:** 0.2**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-speed'**,**

neatName**:** 'Speed'**,**

desc**:** 'The speed that an ant can move'**,**

step**:** 0.01**,**

healthModifier**:** 1000**,**

defaultValue**:** 0.1**,**

value**:** 0.1**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

jawStrength**:** **{**

min**:** 0**,**

max**:** 25**,**

type**:** VALUE\_TYPE**.**integerValue**,**

id**:** 'char-jawStrength'**,**

neatName**:** 'Jaw Strength'**,**

desc**:** 'The stength of the ants jaw (determins how much food the ant can carry)'**,**

step**:** 1**,**

healthModifier**:** 5**,**

defaultValue**:** 15**,**

value**:** 15**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

jawSize**:** **{**

min**:** 0**,**

max**:** 5**,**

type**:** VALUE\_TYPE**.**integerValue**,**

id**:** 'char-jawSize'**,**

neatName**:** 'Jaw Size'**,**

desc**:** 'The amount of damage a soldier ant does when attack'**,**

step**:** 1**,**

healthModifier**:** 30**,**

defaultValue**:** 1**,**

value**:** 1**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

stingSize**:** **{**

min**:** 0**,**

max**:** 5**,**

type**:** VALUE\_TYPE**.**integerValue**,**

id**:** 'char-stingSize'**,**

neatName**:** 'Sting Size'**,**

desc**:** 'The range which a soldier ant can attack'**,**

step**:** 1**,**

healthModifier**:** 10**,**

defaultValue**:** 1**,**

value**:** 1**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

eyesight**:** **{**

min**:** 0**,**

max**:** 10**,**

type**:** VALUE\_TYPE**.**integerValue**,**

id**:** 'char-eyeSight'**,**

neatName**:** 'Eye Sight'**,**

desc**:** 'The range of cells an ant can see around it'**,**

step**:** 1**,**

healthModifier**:** 15**,**

defaultValue**:** 3**,**

value**:** 3**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

eyeAngle**:** **{**

min**:** 0**,**

max**:** Math**.**PI **\*** 2**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-eyeAngle'**,**

neatName**:** 'Eye Angle'**,**

desc**:** 'The angle at which an ant can see'**,**

step**:** 0.01**,**

healthModifier**:** 30**,**

defaultValue**:** Math**.**PI **/** 3**,**

value**:** Math**.**PI **/** 3**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

antennaSize**:** **{**

min**:** 0**,**

max**:** 10**,**

type**:** VALUE\_TYPE**.**integerValue**,**

id**:** 'char-antennaSize'**,**

neatName**:** 'Antenna Size'**,**

desc**:** 'The angle at which an ant can detect pheromones'**,**

step**:** 1**,**

healthModifier**:** 3**,**

defaultValue**:** 3**,**

value**:** 5**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

antennaAngle**:** **{**

min**:** 0**,**

max**:** Math**.**PI **\*** 2**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-antennaAngle'**,**

neatName**:** 'Antenna Angle'**,**

desc**:** 'The range of cells an ant can detect pheromones'**,**

step**:** 0.01**,**

healthModifier**:** 30**,**

defaultValue**:** Math**.**PI **/** 3**,**

value**:** Math**.**PI **/** 3**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

pheromoneConcentration**:** **{**

min**:** 0**,**

max**:** 2**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-pheromoneConcentration'**,**

neatName**:** 'Pheromone Concentration'**,**

desc**:** 'The concentration of pheromones an ant can secrete'**,**

step**:** 0.01**,**

healthModifier**:** 25**,**

defaultValue**:** 0.4**,**

value**:** 0.4**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

nestCoordMemory**:** **{**

min**:** 0**,**

max**:** 1**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-nestCoordMemory'**,**

neatName**:** 'Nest Position Memory'**,**

desc**:** 'A measure of how well the ant knows where the nest is, used when navigating \

to the nest, represents memory of familiarly landmarks near the nest'**,**

step**:** 0.01**,**

healthModifier**:** 500**,**

defaultValue**:** 0.05**,**

value**:** 0.05**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

explorativeInfluence**:** **{**

min**:** 0**,**

max**:** 1**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-explorativeInfluence'**,**

neatName**:** 'Explorative Influence'**,**

desc**:** 'The likelihood of an ant changing direction rather then continue going in the \

direction its facing'**,**

step**:** 0.001**,**

healthModifier**:** 0**,**

defaultValue**:** 0.005**,**

value**:** 0.005**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

pheromoneInfluence**:** **{**

min**:** 0**,**

max**:** 1**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-pheromoneInfluence'**,**

neatName**:** 'Pheromone Influence'**,**

desc**:** 'How likely it is that an ant will follow a pheromones'**,**

step**:** 0.01**,**

healthModifier**:** 0**,**

defaultValue**:** 0.1**,**

value**:** 0.1**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

reproductionWorkerProb**:** **{**

min**:** 0**,**

max**:** 1**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-reproductionWorkerProb'**,**

neatName**:** 'Worker ant probability'**,**

desc**:** 'The probability of a worker ant being born compared with other types of ants'**,**

step**:** 0.05**,**

healthModifier**:** 0**,**

defaultValue**:** 0.5**,**

value**:** 0.5**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

reproductionWorkerFoodCost**:** **{**

min**:** 0**,**

max**:** 50**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-reproductionWorkerFoodCost'**,**

neatName**:** 'Worker ant food cost'**,**

desc**:** 'The amount of food required to create a worker ant (The ant starts with this \

amount of health)'**,**

step**:** 0.1**,**

healthModifier**:** 0**,**

defaultValue**:** 5**,**

value**:** 5**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

reproductionSoldierProb**:** **{**

min**:** 0**,**

max**:** 1**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-reproductionSoldierProb'**,**

neatName**:** 'Soldier ant probability'**,**

desc**:** 'The probability of a soldier ant being born compared with other types of ants'**,**

step**:** 0.05**,**

healthModifier**:** 0**,**

defaultValue**:** 0.05**,**

value**:** 0.05**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

reproductionSoldierFoodCost**:** **{**

min**:** 0**,**

max**:** 50**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-reproductionSoldierFoodCost'**,**

neatName**:** 'Soldier ant food cost'**,**

desc**:** 'The amount of food required to create a soldier ant (The ant starts with this \

amount of health)'**,**

step**:** 0.1**,**

healthModifier**:** 0**,**

defaultValue**:** 8**,**

value**:** 8**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

reproductionQueenProb**:** **{**

min**:** 0**,**

max**:** 1**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-reproductionQueenProb'**,**

neatName**:** 'Queen ant probability'**,**

desc**:** 'The probability of a queen ant being born compared with other types of ants'**,**

step**:** 0.05**,**

healthModifier**:** 0**,**

defaultValue**:** 0.1**,**

value**:** 0.1**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

reproductionQueenFoodCost**:** **{**

min**:** 0**,**

max**:** 50**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-reproductionQueenFoodCost'**,**

neatName**:** 'Queen ant food cost'**,**

desc**:** 'The amount of food required to create a queen ant (The ant starts with this \

amount of health)'**,**

step**:** 0.1**,**

healthModifier**:** 0**,**

defaultValue**:** 15**,**

value**:** 15**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

queenStepsMin**:** **{**

min**:** 0**,**

max**:** 2000**,**

type**:** VALUE\_TYPE**.**integerValue**,**

id**:** 'char-queenStepsMin'**,**

neatName**:** 'Minimum number of Queen steps'**,**

desc**:** 'The minimum number of steps a queen will take until it reaches its new nest site'**,**

step**:** 1**,**

healthModifier**:** 0**,**

defaultValue**:** 400**,**

value**:** 200**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

queenStepsMax**:** **{**

min**:** 0**,**

max**:** 2000**,**

type**:** VALUE\_TYPE**.**integerValue**,**

id**:** 'char-queenStepsMax'**,**

neatName**:** 'Maximum number of Queen steps'**,**

desc**:** 'The maximum number of steps a queen will take until it reaches its new nest site'**,**

step**:** 1**,**

healthModifier**:** 0**,**

defaultValue**:** 800**,**

value**:** 800**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**},**

reproductionRate**:** **{**

min**:** 0**,**

max**:** 0.2**,**

type**:** VALUE\_TYPE**.**floatValue**,**

id**:** 'char-reproductionRate'**,**

neatName**:** 'Reproduction Rate'**,**

desc**:** 'The chance each tick of a new ant being born'**,**

step**:** 0.0001**,**

healthModifier**:** 0**,**

defaultValue**:** 0.01**,**

value**:** 0.01**,**

editable**:** ***true*,**

inputType**:** INPUT\_TYPE**.**slider

**}**

**};**

## core/main.js

// Define file scope variables

***var*** canvasDOM**;**

***var*** canvasCTX**;**

***var*** simulationFoodSystem**;**

***var*** lastLoop **=** ***new*** Date**();**

/\*\*

\* Actions required each tick

\*/

***function*** tick**()** **{**

***var*** thisLoop **=** ***new*** Date**();**

drawMap**(**canvasCTX**);**

updateSpeciesData**();**

// Calculate frame rate

***var*** tickTime **=** TICK **-** **(**thisLoop **-** lastLoop**);**

tickTime **=** **(**tickTime **<=** 0**)** **?** 0 **:** tickTime**;**

setTimeout**(**tick**,** tickTime**);**

CURRENT\_TICK**++;**

lastLoop **=** thisLoop**;** // used for calculating frame rate

**}**

/\*\*

\* Draw all objects onto canvas

\*/

***function*** drawMap**(**ctx**)** **{**

clearCanvas**(**canvasCTX**);**

drawBackground**(**canvasCTX**);**

// Update all of the ants in the sytem

***if*** **(**RUNNING**)** **{**

TICK **+=** 1**;**

***for*** **(*var*** i **=** 0**;** i **<** ANTS\_LIST**.**length**;** i**++)**

ANTS\_LIST**[**i**].**update**();**

// Grow food

simulationFoodSystem**.**growFood**();**

**}**

// Update each cell on the map

***for*** **(*var*** i **=** 0**;** i **<** NUM\_OF\_CELLS**;** i**++)** **{**

***if*** **(**RUNNING**)** **{**

// Update and draw the pheromones in the cell

***for*** **(*var*** k **=** 0**;** k **<** MAP**[**i**].**pheromone**.**length**;** k**++)** **{**

***var*** pheromone **=** MAP**[**i**].**pheromone**[**k**];**

pheromone**.**update**();**

**}**

**}**

// Draw only the first ant

***if*** **(**MAP**[**i**].**ant**.**length **>** 0**)** **{**

***var*** ant **=** MAP**[**i**].**ant**[**0**];**

ant**.**draw**(**ctx**);**

**}** ***else*** ***if*** **(**MAP**[**i**].**pheromone**.**length **>** 0**)** **{**

***var*** pheromone **=** MAP**[**i**].**pheromone**[**0**];**

pheromone**.**draw**(**ctx**);**

**}** ***else*** ***if*** **(**MAP**[**i**].**food **!==** ***void*(**0**))** **{** // Don't draw food on top of ants

***var*** food **=** MAP**[**i**].**food**;**

food**.**draw**(**ctx**);**

**}**

**}**

drawGrid**(**canvasCTX**);**

**}**

//

/\*\*

\* Setups the environment when starting or restarting the simulation

\*/

***function*** createEnviroment**()** **{**

CURRENT\_ID **=** 0**;**

SPECIES\_LIST **=** **[];**

ANTS\_LIST **=** **[];**

RUNNING **=** ***true*;**

// Remove previous data about old species

getElement**(**'data'**).**innerHTML **=** ''**;**

// Create empty map

createMap**();**

// Create food system & add food

simulationFoodSystem **=** ***new*** FoodSystem**();**

simulationFoodSystem**.**addFood**();**

// Create the users species

USER\_SPECIES **=** ***new*** Species**(**genID**());**

USER\_SPECIES**.**colour **=** **{**

worker**:** '#1C1C1C'**,**

soldier**:** '#1C1C1C'**,**

queen**:** '#1C1C1C'**,**

nest**:** '#1C1C1C'**,**

pheromone**:** '#1C1C1C'**,**

**};**

SELECTED\_SPECIES **=** USER\_SPECIES**;**

SPECIES\_LIST**.**push**(**USER\_SPECIES**);**

createSpeciesData**(**USER\_SPECIES**);**

updateUserSpecies**();**

// Add starting queen ants

***for*** **(*var*** i **=** 0**;** i **<** STARTING\_QUEEN\_ANT\_NUMBER**;** i**++)** **{**

***var*** x **=** randInt**({** // random x-coordinate

min**:** 0**,**

max**:** GRID\_SIZE**.**width **-** 1

**});** // -1 as randInt is inclusive

***var*** y **=** randInt**({** // random y-coordinate

min**:** 0**,**

max**:** GRID\_SIZE**.**height **-** 1

**});**

***var*** ant **=** ***new*** Queen**(**genID**(),** **{**

x**:** x**,**

y**:** y

**});**

ant**.**addToMap**();**

ant**.**species **=** USER\_SPECIES**;**

ant**.**colour **=** USER\_SPECIES**.**colour**.**worker**;**

ant**.**species**.**ants**.**push**(**ant**);**

ant**.**health **=** 3000**;** // The starting ant health

ANTS\_LIST**.**push**(**ant**);**

// Start centred with the ant

CANVAS\_OFFSET**.**x **=** **-**x **\*** CELL\_SIZE**.**width **+** CANVAS**.**width **/** 2**;**

CANVAS\_OFFSET**.**y **=** **-**y **\*** CELL\_SIZE**.**height **+** CANVAS**.**height **/** 2**;**

**}**

**}**

// Object holding the functions each button performs

***var*** BUTTONS **=** **{**

reset**:** updateDefaultValues**,**

random**:** updateRandomValues**,**

run**:** runPauseButton**,**

update**:** updateUserSpecies**,**

step**:** step**,**

restart**:** createEnviroment

**};**

***var*** draging **=** ***false*;** // Holds state of map being draged or not

// Once the HTML is loaded

window**.**onload **=** ***function*** **()** **{**

// Get canvas DOM & canvas context

canvasDOM **=** getElement**(**CANVAS**.**name**);**

canvasCTX **=** canvasDOM**.**getContext**(**'2d'**);**

resizeElement**(**canvasDOM**,** CANVAS**);**

// Add event listeners for key press

window**.**onkeydown **=** ***function*** **(**e**)** **{**

e **=** e **||** window**.**event**;**

***var*** charCode **=** e**.**keyCode **||** e**.**which**;**

***switch*** **(**charCode**)** **{**

***case*** LEFT\_ARROW\_KEY**:** // Pan left

CANVAS\_OFFSET**.**x **+=** PAN\_AMOUNT**;**

***break*;**

***case*** RIGHT\_ARROW\_KEY**:** // Pan right

CANVAS\_OFFSET**.**x **-=** PAN\_AMOUNT**;**

***break*;**

***case*** UP\_ARROW\_KEY**:** // Pan up

CANVAS\_OFFSET**.**y **+=** PAN\_AMOUNT**;**

***break*;**

***case*** DOWN\_ARROW\_KEY**:** // Pan down

CANVAS\_OFFSET**.**y **-=** PAN\_AMOUNT**;**

***break*;**

***case*** PLUS\_KEY**:** // Zoom in

zoom**(**ZOOM\_AMOUNT**);**

***break*;**

***case*** MINUS\_KEY**:** // Zoom out

zoom**(-**1 **\*** ZOOM\_AMOUNT**);**

***break*;**

***case*** SPACE\_BAR\_KEY**:** // Pause

runPauseButton**(**getElement**(**'button-run'**));**

e**.**preventDefault**();**

***break*;**

***case*** R\_KEY**:** // restart

createEnviroment**();**

***break*;**

***case*** S\_KEY**:** // step

step**(**getElement**(**'button-step'**));**

**}**

**};**

// The following are two event listeners which watch the mouse wheel they

// calculate the amount the wheel is scrolled to determine amount to zoom.

// Chrome, IE

canvasDOM**.**addEventListener**(**'mousewheel'**,** ***function*** **(**e**)** **{**

// cross-browser wheel delta

***var*** e **=** window**.**event **||** e**;** // old IE support

***var*** delta **=** Math**.**max**(-**1**,** Math**.**min**(**1**,** **(**e**.**wheelDelta **||** **-**e**.**detail**)));**

zoom**(**delta **\*** ZOOM\_AMOUNT**);**

e**.**preventDefault**();** // prevent scrolling the page while zooming

**},** ***false*);**

// FireFox

canvasDOM**.**addEventListener**(**'DOMMouseScroll'**,** ***function*** **(**e**)** **{**

// cross-browser wheel delta

***var*** e **=** window**.**event **||** e**;** // old IE support

***var*** delta **=** Math**.**max**(-**1**,** Math**.**min**(**1**,** **(**e**.**wheelDelta **||** **-**e**.**detail**)));**

zoom**(**delta **\*** ZOOM\_AMOUNT**);**

e**.**preventDefault**();** // prevent scrolling the page while zooming

**},** ***false*);**

// Event listener to watch primary mouse button down

canvasDOM**.**onmousedown **=** ***function*** **(**e**)** **{**

draging **=** ***true*;**

start **=** **{**

x**:** e**.**pageX**,**

y**:** e**.**pageY

**};**

**};**

// Event listener to watch primary touch event

canvasDOM**.**touchstart **=** ***function*** **(**e**)** **{**

draging **=** ***true*;**

start **=** **{**

x**:** e**.**pageX**,**

y**:** e**.**pageY

**};**

**};**

// The starting top left corner of the simulation

***var*** start **=** **{**

x**:** 250**,**

y**:** 310

**};**

// The end coordinate of the mouse on the simulation

***var*** end **=** **{**

x**:** 0**,**

y**:** 0

**};**

// Event listener watching when mouse moves. Updates position of simulation

// when the mouse is moved

canvasDOM**.**onmousemove **=** ***function*** **(**e**)** **{**

***if*** **(**draging**)** **{**

end **=** **{**

x**:** e**.**pageX**,**

y**:** e**.**pageY

**};**

CANVAS\_OFFSET**.**x **=** CANVAS\_OFFSET**.**x **+** **(**end**.**x **-** start**.**x**);**

CANVAS\_OFFSET**.**y **=** CANVAS\_OFFSET**.**y **+** **(**end**.**y **-** start**.**y**);**

start **=** **{**

x**:** e**.**pageX**,**

y**:** e**.**pageY

**};**

**}**

**};**

// Event listener watching when touch movement events. Updates position of

// simulation when a finger is moved across the simulation

canvasDOM**.**touchmove **=** ***function*** **(**e**)** **{**

***if*** **(**draging**)** **{**

end **=** **{**

x**:** e**.**pageX**,**

y**:** e**.**pageY

**};**

CANVAS\_OFFSET**.**x **=** CANVAS\_OFFSET**.**x **+** **(**end**.**x **-** start**.**x**);**

CANVAS\_OFFSET**.**y **=** CANVAS\_OFFSET**.**y **+** **(**end**.**y **-** start**.**y**);**

start **=** **{**

x**:** e**.**pageX**,**

y**:** e**.**pageY

**};**

**}**

**};**

// Event listener watching mouse up event

window**.**onmouseup **=** ***function*** **(**e**)** **{**

draging **=** ***false*;**

**};**

// Event listener watching finger up event

window**.**touchend **=** ***function*** **(**e**)** **{**

draging **=** ***false*;**

**};**

createEnviroment**();**

// Set the sliders to their default values

createCharacteristicInputs**();**

tick**();**

**};**

# Appendix D: Test source code

Ant.atNest.test

// ---------- Ant.atNest ----------

/\*\*

\* Test case - Test that ant is on nest or not

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* A single ant in a static position with a single nest

\*

\* Test data - not on the nest [functional value]

\* Expected - returns false

\*

\* Test data - On the nest [functional value]

\* Expected - returns true

\*/

var AntAtNestTest = new testCase('Test that ant is on nest or not');

var testAntSpecies = {chars : {eyesight : 10, eyeAngle : Math.PI/2}, itemsInView : {ants : [], food : []}}; // simulate a species

var testAntNest = {type : ANT\_TYPE.nest, nest : 0}; // simulate a nest

var testAnt = new Ant(1, {x : 25, y : 25});

testAnt.species = testAntSpecies;

testAnt.direction = 3.67;

testAnt.nest = 0;

AntAtNestTest.callwith = testAnt;

AntAtNestTest.autorun = true;

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

createMap();

// Simulate the dummy nest

MAP[coordToIndex({x : 5, y : 2})].ant = [testAntNest];

AntAtNestTest.createTest(testAnt.atNest, [], 'equal', false);

createMap();

// Simulate the dummy ants

MAP[coordToIndex({x : 25, y : 25})].ant = [testAntNest];

AntAtNestTest.createTest(testAnt.atNest, [], 'equal', true);

AntAtNestTest.summery();

## Ant.scan.test

// ---------- Ant.scan ----------

/\*\*

\* Test case - Test that adds all ants within viewing distance to ant.itemsInView.ants

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* A single ant in a static position with multiple ants around it

\*

\* Test data - No ants in view [edge case]

\* Expected - this.itemsInView.ants.length = 0 i.e. cannot see any ants in view

\*

\* Test data - Multiple ants within viewing distance [functional value]

\* Expected - this.itemsInView.ants.length = 4

\*/

var AntScanVisibleAntsTest = new testCase('Test that adds all ants within viewing distance to ant.itemsInView.ants');

var testAntSpecies = {chars : {eyesight : 10, eyeAngle : Math.PI/2}}; // simulate a species

var testAnt = new Ant(1, {x : 25, y : 25});

testAnt.species = testAntSpecies;

testAnt.direction = 3.67;

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

createMap();

// Simulate the dummy ants

MAP[coordToIndex({x : 18, y : 34})].ant = ['ant1'];

MAP[coordToIndex({x : 25, y : 22})].ant = ['ant2'];

MAP[coordToIndex({x : 30, y : 32})].ant = ['ant3', 'ant4'];

testAnt.scan();

AntScanVisibleAntsTest.createTest(testAnt.itemsInView.ants.length, [], 'equal', 0);

createMap();

// Simulate the dummy ants

MAP[coordToIndex({x : 26, y : 32})].ant = ['ant1'];

MAP[coordToIndex({x : 26, y : 29})].ant = ['ant2'];

MAP[coordToIndex({x : 23, y : 28})].ant = ['ant3', 'ant4'];

testAnt.scan();

AntScanVisibleAntsTest.createTest(testAnt.itemsInView.ants.length, [], 'equal', 4);

AntScanVisibleAntsTest.testAll();

AntScanVisibleAntsTest.summery();

/\*\*

\* Test case - Test that adds all food within viewing distance to ant.itemsInView.food

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* A single ant in a static position with multiple pieces of food around it

\*

\* Test data - No food in view [edge case]

\* Expected - this.itemsInView.food.length = 0 i.e. cannot see any food in view

\*/

var AntScanVisibleAntsTest = new testCase('Test that adds all food within viewing distance to ant.itemsInView.food');

var testAntSpecies = {chars : {eyesight : 10, eyeAngle : Math.PI/2}}; // simulate a species

var testAnt = new Ant(1, {x : 25, y : 25});

testAnt.species = testAntSpecies;

testAnt.direction = 3.67;

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

createMap();

// Simulate the dummy pieces of food

MAP[coordToIndex({x : 18, y : 34})].food = 'food1';

MAP[coordToIndex({x : 25, y : 22})].food = 'food2';

MAP[coordToIndex({x : 30, y : 32})].food = 'food3';

testAnt.scan();

AntScanVisibleAntsTest.createTest(testAnt.itemsInView.food.length, [], 'equal', 0);

createMap();

// Simulate the dummy pieces of food

MAP[coordToIndex({x : 26, y : 32})].food = 'food1';

MAP[coordToIndex({x : 26, y : 29})].food = 'food2';

MAP[coordToIndex({x : 23, y : 28})].food = 'food3';

testAnt.scan();

AntScanVisibleAntsTest.createTest(testAnt.itemsInView.food.length, [], 'equal', 3);

AntScanVisibleAntsTest.testAll();

AntScanVisibleAntsTest.summery();

## Ant.secrete.test

// ---------- Ant.secrete ----------

/\*\*

\* Test case - Test that adds pheromone of correct concentration

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* MAX\_PHEROMONE\_CONCENTRATION = 1;

\* A single ant in a static position with pheromones under it

\*

\* Test data - No pheromones to start [functional value]

\* Expected - A new pheromones of concentration 0.5

\*

\* Test data - A pheromone of the same species (testing adds pheromones) [functional value]

\* Expected - A new pheromones of concentration 0.9 i.e. 0.4 + 0.5

\*

\* Test data - A pheromone of a different species (testing dosn't add pheromones) [functional value]

\* Expected - A new pheromones of concentration 0.5

\*

\* Test data - A pheromone of the same species (testing does not exceed MAX\_PHEROMONE\_CONCENTRATION) [functional value]

\* Expected - A new pheromones of concentration 1

\*/

var AntSecreteTest = new testCase('Test that adds pheromone of correct concentration');

AntSecreteTest.autorun = true;

var testAntSpecies = {chars : {pheromoneConcentration : 0.5}}; // simulate a species

var testAnt = new Ant(1, {x : 25, y : 25});

testAnt.species = testAntSpecies;

testAnt.direction = 3.67;

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

createMap();

testAnt.secrete();

AntSecreteTest.createTest(MAP[coordToIndex({x : 25, y : 25})].pheromone.length, [], 'equal', 1);

AntSecreteTest.createTest(MAP[coordToIndex({x : 25, y : 25})].pheromone[0].concentration, [], 'equal', 0.5);

createMap();

// Simulate the dummy pheromone

MAP[coordToIndex({x : 25, y : 25})].pheromone = [{species : testAntSpecies, concentration : 0.4}];

testAnt.secrete();

AntSecreteTest.createTest(MAP[coordToIndex({x : 25, y : 25})].pheromone.length, [], 'equal', 1);

AntSecreteTest.createTest(MAP[coordToIndex({x : 25, y : 25})].pheromone[0].concentration, [], 'equal', 0.9);

createMap();

// Simulate the dummy pheromone

MAP[coordToIndex({x : 25, y : 25})].pheromone = [{species : 0, concentration : 0.4}];

testAnt.secrete();

AntSecreteTest.createTest(MAP[coordToIndex({x : 25, y : 25})].pheromone.length, [], 'equal', 2);

AntSecreteTest.createTest(MAP[coordToIndex({x : 25, y : 25})].pheromone[1].concentration, [], 'equal', 0.5);

AntSecreteTest.createTest(MAP[coordToIndex({x : 25, y : 25})].pheromone[0].concentration, [], 'equal', 0.4);

createMap();

// Simulate the dummy pheromone

MAP[coordToIndex({x : 25, y : 25})].pheromone = [{species : testAntSpecies, concentration : 0.8}];

testAnt.secrete();

AntSecreteTest.createTest(MAP[coordToIndex({x : 25, y : 25})].pheromone.length, [], 'equal', 1);

AntSecreteTest.createTest(MAP[coordToIndex({x : 25, y : 25})].pheromone[0].concentration, [], 'equal', MAX\_PHEROMONE\_CONCENTRATION);

AntSecreteTest.summery();

## Ant.seeNest.test

// ---------- Ant.seeNest ----------

/\*\*

\* Test case - Test that can see nest when in range

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* A single ant in a static position with a single nest

\*

\* Test data - Cannot see the nest [functional value]

\* Expected - returns false

\*

\* Test data - An ant within the ants view [functional value]

\* Expected - returns true

\*/

var AntSeeNestTest = new testCase('Test that can see nest when in range');

var testAntSpecies = {chars : {eyesight : 10, eyeAngle : Math.PI/2}, itemsInView : {ants : [], food : []}}; // simulate a species

var testAntNest = {type : ANT\_TYPE.nest, nest : 0}; // simulate a nest

var testAnt = new Ant(1, {x : 25, y : 25});

testAnt.species = testAntSpecies;

testAnt.direction = 3.67;

testAnt.nest = 0;

AntSeeNestTest.callwith = testAnt;

AntSeeNestTest.autorun = true;

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

createMap();

// Simulate the dummy ants

MAP[coordToIndex({x : 5, y : 2})].ant = [testAntNest];

testAnt.scan();

AntSeeNestTest.createTest(testAnt.seeNest, [], 'equal', false);

createMap();

// Simulate the dummy ants

MAP[coordToIndex({x : 23, y : 26})].ant = [testAntNest];

testAnt.scan();

AntSeeNestTest.createTest(testAnt.seeNest, [], 'equal', true);

AntSeeNestTest.summery();

## Ant.smell.test

// ---------- Ant.smell ----------

/\*\*

\* Test case - Test that adds all pheromones within range to pheromonesInRange

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\* A single ant in a static position with multiple ants around it

\*

\* Test data - No pheromones in range [functional value]

\* Expected - this.pheromonesInRange.length = 0 i.e. cannot smell any pheromones in view

\*

\* Test data - 4 Pheromones in range with one ontop of the ant [functional value]

\* Expected - this.pheromonesInRange.length = 3

\*/

var AntSmellTest = new testCase('Test that adds all pheromones within range to pheromonesInRange');

AntSmellTest.autorun = true;

var testAntSpecies = {chars : {antennaSize : 10, antennaAngle : Math.PI/2}}; // simulate a species

var testAnt = new Ant(1, {x : 25, y : 25});

testAnt.species = testAntSpecies;

testAnt.direction = 3.67;

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

createMap();

// Simulate the dummy pheromones

MAP[coordToIndex({x : 10, y : 10})].pheromone = ['pheromone1'];

MAP[coordToIndex({x : 10, y : 15})].pheromone = ['pheromone2'];

MAP[coordToIndex({x : 12, y : 10})].pheromone = ['pheromone3', 'pheromone4'];

testAnt.smell();

AntSmellTest.createTest(testAnt.pheromonesInRange.length, [], 'equal', 0);

createMap();

// Simulate the dummy pheromones

MAP[coordToIndex({x : 25, y : 25})].pheromone = ['pheromone1'];

MAP[coordToIndex({x : 23, y : 26})].pheromone = ['pheromone2'];

MAP[coordToIndex({x : 22, y : 27})].pheromone = ['pheromone3', 'pheromone4'];

testAnt.smell();

AntSmellTest.createTest(testAnt.pheromonesInRange.length, [], 'equal', 3);

AntSmellTest.summery();

## Ant.takeFood.test

// ---------- Ant.takeFood ----------

/\*\*

\* Test case - Test ant takes correct amount of food

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* ANT\_FOOD\_TAKE\_SPEED = 1;

\* A single ant in a static position with piece of food underneith it

\*

\* Test data - void(0) i.e. No food [functional value]

\* Expected - return 0

\*/

var AntTakeFoodTest = new testCase('Test ant takes correct amount of food');

AntTakeFoodTest.autorun = true;

var testAntSpecies = {chars : {pheromoneConcentration : 0.5}}; // simulate a species

var testAnt = new Ant(1, {x : 25, y : 25});

testAnt.species = testAntSpecies;

testAnt.direction = 3.67;

AntTakeFoodTest.callwith = testAnt;

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

ANT\_FOOD\_TAKE\_SPEED = 1;

createMap();

AntTakeFoodTest.createTest(testAnt.takeFood, [void(0)], 'equal', 0);

createMap();

// Simulate the dummy food

MAP[coordToIndex({x : 25, y : 25})].food = new Food(void(0), 3, {x : 25, y : 25});

AntTakeFoodTest.createTest(testAnt.takeFood, [MAP[coordToIndex({x : 25, y : 25})].food], 'equal', 1);

AntTakeFoodTest.createTest(MAP[coordToIndex({x : 25, y : 25})].food.amount, [], 'equal', 2);

AntTakeFoodTest.createTest(testAnt.takeFood, [MAP[coordToIndex({x : 25, y : 25})].food], 'equal', 1);

AntTakeFoodTest.createTest(MAP[coordToIndex({x : 25, y : 25})].food.amount, [], 'equal', 1);

AntTakeFoodTest.createTest(testAnt.takeFood, [MAP[coordToIndex({x : 25, y : 25})].food], 'equal', 1);

AntTakeFoodTest.createTest(MAP[coordToIndex({x : 25, y : 25})].food, [], 'equal', void(0));

AntTakeFoodTest.createTest(testAnt.takeFood, [MAP[coordToIndex({x : 25, y : 25})].food], 'equal', 0);

AntTakeFoodTest.summery();

/\* Previously failed:

OLD FUNCTION

Ant.prototype.takeFood = function(food) {

if (this.isFood(food)) { // If food

food.amount -= 1; // Take a single piece of food

this.sleep += ANT\_FOOD\_TAKE\_SPEED;

if (this.isFood(food)) // If food is all gone remove it from the map

food.removeFromMap();

return 1;

} else {

return 0;

}

};

NEW FUNCTION - added a not on the line if (!this.isFood(food)) as was only taking a single piece befor

Ant.prototype.takeFood = function(food) {

if (this.isFood(food)) { // If food

food.amount -= 1; // Take a single piece of food

this.sleep += ANT\_FOOD\_TAKE\_SPEED;

if (!this.isFood(food)) // If food is all gone remove it from the map

food.removeFromMap();

return 1;

} else {

return 0;

}

};

\*/

## Ant.wonder.test

// ---------- Ant.wonder ----------

function arrowTo(coord1, angle, blocks) {

window.onload = function () {

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

drawRect(ctx, scaleCoord(coord1), CELL\_SIZE, '#FF0000');

for (var i = 0; i < blocks.length; i++) {

drawRect(ctx, scaleCoord(blocks[i]), CELL\_SIZE, '#0000FF');

}

ctx.save();

var scaledCoord = scaleCoord(coord1);

// Translate and rotate the canvas (done so can draw at an angle)

ctx.translate(scaledCoord.x + CELL\_SIZE.width/2, scaledCoord.y + CELL\_SIZE.height/2);

ctx.rotate(angle);

drawLine(ctx, {x : 0, y : 0}, {x : 0, y : -100}, '#FF0000', 1);

drawLine(ctx, {x : 0, y : -100}, {x : 5, y : -95}, '#FF0000', 1);

drawLine(ctx, {x : 0, y : -100}, {x : -5, y : -95}, '#FF0000', 1);

ctx.restore();

}

}

/\*\*

\* Test case - Test that ant picks correct direction given some amount of pheromones

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* MAX\_PHEROMONE\_CONCENTRATION = 1;

\* A single ant in a static position with pheromones around it

\*

\* Test data - No pheromones in view [functional value]

\* Expected - this.direction === this.prioritizeDirection

\*

\* Test data - A single pheromone [functional value]

\* Expected - Ant is pointing at the pheromone

\*

\* Test data - Multiple pheromones [functional value]

\* Expected - Ant is pointing slightly upwards to the right

\*/

var AntWonderDirectionTest = new testCase('Test that ant picks correct direction given some amount of pheromones');

AntWonderDirectionTest.autorun = true;

var testAntSpecies = {chars : {antennaSize : 10, antennaAngle : 2\*Math.PI, exploitativeness : 0, pheromoneInfluence : 1}} // simulate a species

var testAnt = new Ant(1, {x : 25, y : 25});

testAnt.species = testAntSpecies;

testAnt.direction = Math.PI \* 2 - Math.PI / 4;

AntWonderDirectionTest.callwith = testAnt;

AntWonderDirectionTest.autorun = true;

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

MAX\_PHEROMONE\_CONCENTRATION = 1;

createMap();

testAnt.smell();

testAnt.wonder();

AntWonderDirectionTest.createTest(testAnt.direction, [], 'equal', testAnt.prioritizeDirection);

arrowTo({x : 25, y : 25}, testAnt.direction, []);

// 2

testAnt.direction = Math.PI / 2;

createMap();

// Simulate the dummy pheromones

MAP[coordToIndex({x : 27, y : 22})].pheromone = [{coord : {x : 27, y : 22}, species : testAntSpecies, concentration : 0.4}];

testAnt.smell();

testAnt.wonder();

AntWonderDirectionTest.createTest(testAnt.direction, [], 'desc', 'Ant points in the direction of the pheromone');

arrowTo({x : 25, y : 25}, testAnt.direction, [{x : 27, y : 22}]);

// 3

testAnt.direction = Math.PI \* 2 - Math.PI / 4;

createMap();

// Simulate the dummy pheromones

MAP[coordToIndex({x : 18, y : 22})].pheromone = [{coord : {x : 18, y : 22}, species : testAntSpecies, concentration : 0.4}];

MAP[coordToIndex({x : 18, y : 20})].pheromone = [{coord : {x : 18, y : 20}, species : testAntSpecies, concentration : 0.4}];

MAP[coordToIndex({x : 18, y : 28})].pheromone = [{coord : {x : 18, y : 28}, species : testAntSpecies, concentration : 0.4}];

testAnt.smell();

testAnt.wonder();

AntWonderDirectionTest.createTest(testAnt.direction, [], 'desc', 'Ant points in the direction of the pheromone');

arrowTo({x : 25, y : 25}, testAnt.direction, [{x : 18, y : 22}, {x : 18, y : 20}, {x : 18, y : 28}]);

AntWonderDirectionTest.summery();

## Food.draw.test

// ---------- Food.draw ----------

/\*\*

\* Test case - Test draw draws a piece of food correctly

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\*

\* Test data - A single piece of food of amount 1

\* Expected - A piece of food of alpha value 0.2

\*

\* Test data - A single piece of food of amount 5

\* Expected - A piece of food of alpha value 1

\*

\* Test data - A single piece of food of amount 0

\* Expected - A piece of food of alpha value 0

\*/

window.onload = function() {

var FoodDrawTest = new testCase('Test draw draws a piece of food correctly');

var foodSystem = {colour : '#00FF00', variation : {max : 5}};

FoodDrawTest.autorun = true;

CANVAS\_OFFSET = {x : 0, y : 0};

MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

CELL\_SIZE = {width : 50, height : 50};

GRID\_SIZE = {width : 10, height : 10};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

var testFood = new Food(foodSystem, 1, {x : 5, y : 5});

testFood.size = CELL\_SIZE;

testFood.amount = 1;

FoodDrawTest.callwith = testFood;

FoodDrawTest.createTest(testFood.draw, [ctx], 'desc', 'A Pheromone with 0.5 alpha value');

// 2

testFood.amount = 5;

FoodDrawTest.createTest(testFood.draw, [ctx], 'desc', 'No visible pheromones');

// 3

testFood.amount = 0;

FoodDrawTest.createTest(testFood.draw, [ctx], 'desc', 'A Pheromone with 1 alpha value');

FoodDrawTest.summery();

}

## FoodSystem.addFood.test

// ---------- FoodSystem.addFood ----------

function drawFood(ctx) {

for (var i = 0; i < NUM\_OF\_CELLS; i++) {

if (MAP[i].food !== void(0))

MAP[i].food.draw(ctx);

}

}

/\*\*

\* Test case - Test that addFood adds random looking food all over the map

\*

\* Environment:

\* CANVAS\_OFFSET = {x : 0, y : 0};

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\*

\* Test data - Run three times

\* Expected - Random looking placement of food throughout the map

\*/

window.onload = function() {

var addFoodTest = new testCase('Test that addFood adds random looking food all over the map');

addFoodTest.autorun = true;

CANVAS\_OFFSET = {x : 0, y : 0};

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

FOOD\_CHANCE = 0.1;

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

var testFoodSystem = new FoodSystem();

addFoodTest.callwith = testFoodSystem;

// 1

createMap();

addFoodTest.createTest(testFoodSystem.addFood, [], 'desc', 'Random looking placement of food throughout the map');

drawFood(ctx);

// 2

createMap();

addFoodTest.createTest(testFoodSystem.addFood, [], 'desc', 'Random looking placement of food throughout the map');

drawFood(ctx);

// 3

createMap();

addFoodTest.createTest(testFoodSystem.addFood, [], 'desc', 'Random looking placement of food throughout the map');

drawFood(ctx);

addFoodTest.summery();

}

## FoodSystem.addFoodBlob.test

// ---------- FoodSystem.addFoodBlob ----------

function drawFood(ctx) {

for (var i = 0; i < NUM\_OF\_CELLS; i++) {

if (MAP[i].food !== void(0))

MAP[i].food.draw(ctx);

}

}

/\*\*

\* Test case - Test that addFoodBlob adds a food blob of correct size

\*

\* Environment:

\* CANVAS\_OFFSET = {x : 0, y : 0};

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\*

\* Test data - Food blob radius = 5,

\* Expected - A green circle radius 5 which is darkest in the middle and gets brighter the further out it goes

\*

\* Test data - Food blob radius = 25,

\* Expected - A green circle radius 25 which is darkest in the middle and gets brighter the further out it goes

\*

\* Test data - Food blob radius = 0,

\* Expected - A green circle radius 5 which is darkest in the middle and gets brighter the further out it goes

\*/

window.onload = function() {

var addFoodBlobTest = new testCase('Test that addFoodBlob adds a food blob of correct size');

addFoodBlobTest.autorun = true;

CANVAS\_OFFSET = {x : 0, y : 0};

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

FOOD\_CHANCE = 0.1;

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

var testFoodSystem = new FoodSystem();

addFoodBlobTest.callwith = testFoodSystem;

// 1

createMap();

addFoodBlobTest.createTest(testFoodSystem.addFoodBlob, [{x : 25, y : 25}, 5], 'desc', 'A green circle radius 5 which is darkest in the middle and gets brighter the further out it goes');

drawFood(ctx);

// 2

createMap();

addFoodBlobTest.createTest(testFoodSystem.addFoodBlob, [{x : 25, y : 25}, 25], 'desc', 'A green circle radius 25 which is darkest in the middle and gets brighter the further out it goes');

drawFood(ctx);

// 3

createMap();

addFoodBlobTest.createTest(testFoodSystem.addFoodBlob, [{x : 25, y : 25}, 0], 'desc', 'A green circle radius 0 which is darkest in the middle and gets brighter the further out it goes');

drawFood(ctx);

addFoodBlobTest.summery();

}

## Nest.calcSpeciesCost.test

// ---------- Nest.calcSpeciesCost ----------

/\*\*

\* Test case - Test that calcSpeciesCost correctly returns the sum of all characteristics

\*

\* CHARS = {speed : {healthModifier : 100}, stingSize : {healthModifier : 57}, eyesight : {healthModifier : 23}};

\*

\* Test data - species.chars = {speed : 0.51, stingSize : 3, eyesight : 5};

\* Expected - specieCost = 337

\*

\* Test data - species.chars = {speed : 0, stingSize : 0, eyesight : 0};

\* Expected - specieCost = 0

\*/

var calcSpeciesCostTest = new testCase('Test that calcSpeciesCost correctly returns the sum of all characteristics');

calcSpeciesCostTest.autorun = true;

var species = {chars : {speed : 0.51, stingSize : 3, eyesight : 5}};

CHARS = {speed : {healthModifier : 100}, stingSize : {healthModifier : 57}, eyesight : {healthModifier : 23}};

var testNest = new Nest();

calcSpeciesCostTest.callwith = testNest;

testNest.species = species;

// 1

var species = {chars : {speed : 0.51, stingSize : 3, eyesight : 5}};

calcSpeciesCostTest.createTest(testNest.calcSpeciesCost, [], 'equal', 337);

// 2

species = {chars : {speed : 0, stingSize : 0, eyesight : 0}};

testNest.species = species;

calcSpeciesCostTest.createTest(testNest.calcSpeciesCost, [], 'equal', 0);

calcSpeciesCostTest.summery();

## Nest.reproduce.test

// ---------- Nest.reproduce ----------

/\*\*

\* Test case - Test that reproduce produces the correct probabilities

\*

\* Note : not testing Nest.viable function so is redefined to always return true

\*

\* Test data - Soldier probability = 1, worker and queen probabilities = 0

\* Expected - Only soldier ants produced after 1000 runs.

\*

\* Test data - Soldier, Worker and Queen probabilities are all 1

\* Expected - A 1/3 chance of each type of ant

\*

\* Test data - Soldier, Worker and Queen probabilities are all 0

\* Expected - No ants created

\*/

var reproductionProbabilitesTest = new testCase('Test that reproduce produces the correct probabilities');

reproductionProbabilitesTest.autorun = true;

NUMBER\_OF\_ITERATIONS = 10000;

NUMBER\_OF\_WORKERS = 0;

NUMBER\_OF\_SOLDIERS = 0;

NUMBER\_OF\_QUEENS = 0;

var species = {chars : {reproductionWorkerProb : 0, reproductionSoldierProb : 1, reproductionQueenProb : 0}};

var testNest = new Nest();

reproductionProbabilitesTest.callwith = testNest;

testNest.species = species;

testNest.viable = function(antType) {

return true;

}

testNest.createAnt = function(antType) {

switch (antType) {

case ANT\_TYPE.worker:

NUMBER\_OF\_WORKERS += 1;

break;

case ANT\_TYPE.soldier:

NUMBER\_OF\_SOLDIERS += 1;

break;

case ANT\_TYPE.queen:

NUMBER\_OF\_QUEENS += 1;

break;

}

}

// 1

for (var i = 0; i < NUMBER\_OF\_ITERATIONS; i++) {

testNest.reproduce();

}

var totalProb = NUMBER\_OF\_WORKERS + NUMBER\_OF\_SOLDIERS + NUMBER\_OF\_QUEENS;

var workerProb = NUMBER\_OF\_WORKERS / totalProb \* 100;

var soldierProb = NUMBER\_OF\_SOLDIERS / totalProb \* 100;

var queenProb = NUMBER\_OF\_QUEENS / totalProb \* 100;

reproductionProbabilitesTest.createTest(workerProb, [], 'desc', 'The worker probability = 0%');

reproductionProbabilitesTest.createTest(soldierProb, [], 'desc', 'The soldier probability = 100%');

reproductionProbabilitesTest.createTest(queenProb, [], 'desc', 'The queen probability = 0%');

// 2

NUMBER\_OF\_WORKERS = 0;

NUMBER\_OF\_SOLDIERS = 0;

NUMBER\_OF\_QUEENS = 0;

species = {chars : {reproductionWorkerProb : 1, reproductionSoldierProb : 1, reproductionQueenProb : 1}};

testNest.species = species;

for (var i = 0; i < NUMBER\_OF\_ITERATIONS; i++) {

testNest.reproduce();

}

var totalProb = NUMBER\_OF\_WORKERS + NUMBER\_OF\_SOLDIERS + NUMBER\_OF\_QUEENS;

var workerProb = NUMBER\_OF\_WORKERS / totalProb \* 100;

var soldierProb = NUMBER\_OF\_SOLDIERS / totalProb \* 100;

var queenProb = NUMBER\_OF\_QUEENS / totalProb \* 100;

reproductionProbabilitesTest.createTest(workerProb, [], 'desc', 'The worker probability ~= 33.3%');

reproductionProbabilitesTest.createTest(soldierProb, [], 'desc', 'The soldier probability ~= 33.3%');

reproductionProbabilitesTest.createTest(queenProb, [], 'desc', 'The queen probability ~= 33.3%');

// 3

NUMBER\_OF\_WORKERS = 0;

NUMBER\_OF\_SOLDIERS = 0;

NUMBER\_OF\_QUEENS = 0;

species = {chars : {reproductionWorkerProb : 0, reproductionSoldierProb : 0, reproductionQueenProb : 0}};

testNest.species = species;

for (var i = 0; i < NUMBER\_OF\_ITERATIONS; i++) {

testNest.reproduce();

}

reproductionProbabilitesTest.createTest(NUMBER\_OF\_WORKERS, [], 'equal', 0);

reproductionProbabilitesTest.createTest(NUMBER\_OF\_SOLDIERS, [], 'equal', 0);

reproductionProbabilitesTest.createTest(NUMBER\_OF\_QUEENS, [], 'equal', 0);

reproductionProbabilitesTest.summery();

## NestPiece.draw.test

// ---------- NestPiece.draw ----------

/\*\*

\* Test case - Test draw draws the nest piece correctly

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\*

\* Test data - A single nest piece

\* Expected - A black square

\*/

window.onload = function() {

var NestPieceDrawTest = new testCase('Test draw draws the nest piece correctly');

NestPieceDrawTest.autorun = true;

CANVAS\_OFFSET = {x : 0, y : 0};

MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

CELL\_SIZE = {width : 50, height : 50};

GRID\_SIZE = {width : 10, height : 10};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

var species = {colour : {nest : '#000000'}};

var nest = {};

nest.species = species;

var testNestPiece = new NestPiece(1, {x : 5, y : 5}, nest);

testNestPiece.size = CELL\_SIZE;

NestPieceDrawTest.callwith = testNestPiece;

NestPieceDrawTest.createTest(testNestPiece.draw, [ctx], 'desc', 'A black square');

NestPieceDrawTest.summery();

}

## Pheromone.draw.test

// ---------- Pheromone.draw ----------

/\*\*

\* Test case - Test draw draws the pheromone correctly

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\*

\* Test data - A single pheromone of concentration 0.5

\* Expected - A Pheromone with 0.5 alpha value

\*

\* Test data - A single pheromone of concentration 0

\* Expected - No visible pheromones

\*

\* Test data - A single pheromone of concentration 1

\* Expected - A Pheromone with 1 alpha value

\*/

window.onload = function() {

var PheromoneDrawTest = new testCase('Test draw draws the pheromone correctly');

var species = {colour : {pheromone : '#FF0000'}};

PheromoneDrawTest.autorun = true;

CANVAS\_OFFSET = {x : 0, y : 0};

MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

CELL\_SIZE = {width : 50, height : 50};

GRID\_SIZE = {width : 10, height : 10};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

var testPheromone = new Pheromone(1, {x : 5, y : 5});

testPheromone.species = species;

testPheromone.size = CELL\_SIZE;

testPheromone.concentration = 0.5;

PheromoneDrawTest.callwith = testPheromone;

PheromoneDrawTest.createTest(testPheromone.draw, [ctx], 'desc', 'A Pheromone with 0.5 alpha value');

// 2

testPheromone.concentration = 0;

PheromoneDrawTest.createTest(testPheromone.draw, [ctx], 'desc', 'No visible pheromones');

// 3

testPheromone.concentration = 1;

PheromoneDrawTest.createTest(testPheromone.draw, [ctx], 'desc', 'A Pheromone with 1 alpha value');

PheromoneDrawTest.summery();

}

## Queen.draw.test

// ---------- Queen.draw ----------

/\*\*

\* Test case - Test draw draws the queen correctly

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\*

\* Test data - A single queen ant

\* Expected - A black circle

\*/

window.onload = function() {

var QueenDrawTest = new testCase('Test draw draws the queen correctly');

QueenDrawTest.autorun = true;

CANVAS\_OFFSET = {x : 0, y : 0};

MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

CELL\_SIZE = {width : 50, height : 50};

GRID\_SIZE = {width : 10, height : 10};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

var species = {}

var testAnt = new Queen(1, {x : 5, y : 5});

testAnt.direction = 4.104;

testAnt.size = CELL\_SIZE;

QueenDrawTest.callwith = testAnt;

QueenDrawTest.createTest(testAnt.draw, [ctx], 'desc', 'A black circle');

QueenDrawTest.summery();

}

## Soldier.attack.test

// ---------- Solider attack ----------

function tick() {

testAnt.update();

targetAnt.update();

simpleGrid(CTX);

for (var i = 0; i < NUM\_OF\_CELLS; i++) {

if (MAP[i].ant.length > 0) {

MAP[i].ant[0].draw(CTX);

}

}

if (testAnt.targetAnt !== void(0)) {

console.log('Target aquired, following.');

}

if (targetAnt.dead === true) {

console.log('Target ant has died.');

} else {

setTimeout(tick, 10);

console.log('searching for target ant');

}

}

/\*\*

\* Test case - Simulate a soldier ant attacking a worker an from another species

\*

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* A single ant and nest

\*/

var SoldierAttackTest = new testCase('Simulate a soldier ant attacking a worker an from another species');

SoldierAttackTest.autorun = true;

// Configure environment

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

STARTING\_QUEEN\_ANT\_NUMBER = 0;

NEST\_GUARD\_RADIUS = 20;

var CTX;

// Create the soldier ant

var testAnt = new Soldier(genID(), {x : 30, y : 30});

testAnt.species = new Species(); // starts with default values

testAnt.size = CELL\_SIZE; // starts with default values

testAnt.goal = GOAL.guardPheromone; // i.e. just wonder around

// Add target ant

var targetAnt = new Worker(genID(), {x : 5, y : 35});

targetAnt.species = new Species(); // starts with default values

targetAnt.size = CELL\_SIZE; // starts with default values

targetAnt.health = 5000;

createMap();

SoldierAttackTest.createTest('N/A', [], 'desc', 'Watch solidier ant find other ant, follow it and then attack it.');

window.onload = function () {

CTX = getElement(CANVAS.name).getContext('2d');

setTimeout(tick, 10);

}

SoldierAttackTest.summery();

## Soldier.draw.test

// ---------- Soldier.draw ----------

/\*\*

\* Test case - Test draw draws the soldier correctly

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\*

\* Test data - A single soldier ant

\* Expected - A black square with a white cross over it in the centre

\*/

window.onload = function() {

var SoldierDrawTest = new testCase('Test draw draws the soldier correctly');

SoldierDrawTest.autorun = true;

CANVAS\_OFFSET = {x : 0, y : 0};

MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

CELL\_SIZE = {width : 50, height : 50};

GRID\_SIZE = {width : 10, height : 10};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

var testAnt = new Soldier(1, {x : 5, y : 5});

testAnt.direction = 4.104;

testAnt.size = CELL\_SIZE;

SoldierDrawTest.callwith = testAnt;

SoldierDrawTest.createTest(testAnt.draw, [ctx], 'desc', 'A black square with a white cross over it in the centre');

SoldierDrawTest.summery();

}

## Soldier.guardFood.test

// ---------- Solidier guard food ----------

function tick() {

testAnt.update();

simpleGrid(CTX);

for (var i = 0; i < NUM\_OF\_CELLS; i++) {

if (MAP[i].ant.length > 0) {

MAP[i].ant[0].draw(CTX);

} else if (MAP[i].food !== void(0))

MAP[i].food.draw(CTX);

}

if (testAnt.moving === false) {

console.log('Found food and guarding it.');

} else if (testAnt.goal === GOAL.guardFood) {

setTimeout(tick, 10);

console.log('searching for food');

}

}

/\*\*

\* Test case - Simulate a single ant guarding a food supply

\*

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* A single ant and nest

\*/

var SoldierGuardFoodTest = new testCase('Simulate a single ant guarding a food supply');

SoldierGuardFoodTest.autorun = true;

// Configure environment

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

STARTING\_QUEEN\_ANT\_NUMBER = 0;

NEST\_GUARD\_RADIUS = 20;

var CTX;

// Create the ant

var testAnt = new Soldier(genID(), {x : 80, y : 80});

testAnt.species = new Species(); // starts with default values

testAnt.size = CELL\_SIZE; // starts with default values

testAnt.direction = 0;

testAnt.goal = GOAL.guardFood;

createMap();

// Add Food

var fs = new FoodSystem();

fs.variation.max = 1;

MAP[coordToIndex({x : 25, y : 25})].food = new Food(fs, 1, {x : 25, y : 25});

SoldierGuardFoodTest.createTest('N/A', [], 'desc', 'watch ant find food and guard it.');

window.onload = function () {

CTX = getElement(CANVAS.name).getContext('2d');

setTimeout(tick, 10);

}

SoldierGuardFoodTest.summery();

## Soldier.guardNest.test

// ---------- Solidier guard nest ----------

function tick() {

testAnt.update();

simpleGrid(CTX);

for (var i = 0; i < NUM\_OF\_CELLS; i++) {

if (MAP[i].ant.length > 0) {

MAP[i].ant[0].draw(CTX);

}

}

if (testAnt.steps > 0)

console.log('Found nest, going to guard distance');

if (testAnt.moving === false) {

console.log('Guarding.');

} else if (testAnt.goal === GOAL.guardNest) {

setTimeout(tick, 100);

console.log('searching for nest');

}

}

/\*\*

\* Test case - Simulate a single ant guarding its nest

\*

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* A single ant and nest

\*/

var SoldierGuardNestTest = new testCase('Simulate a single ant guarding its nest');

SoldierGuardNestTest.autorun = true;

// Configure environment

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

STARTING\_QUEEN\_ANT\_NUMBER = 0;

NEST\_GUARD\_RADIUS = 20;

var CTX;

// Create the ant

var testAnt = new Soldier(genID(), {x : 80, y : 80});

testAnt.species = new Species(); // starts with default values

testAnt.size = CELL\_SIZE; // starts with default values

testAnt.direction = 0;

testAnt.goal = GOAL.guardNest;

createMap();

// Add Nest

var nest = new Nest(genID(), {x : 25, y : 25});

nest.species = testAnt.species;

testAnt.nest = nest;

nest.createNest();

SoldierGuardNestTest.createTest('N/A', [], 'desc', 'watch ant find nest and guard it.');

window.onload = function () {

CTX = getElement(CANVAS.name).getContext('2d');

setTimeout(tick, 1000);

}

SoldierGuardNestTest.summery();

## Soldier.soldiersInView.test

// ---------- Soldier.soldiersInView ----------

/\*\*

\* Test case - Test that soldier in view returns true when there are soldiers in view

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\*

\* Test data - No ants in view

\* Expected - returns false i.e. no soldiers in view

\*

\* Test data - Multiple ants within viewing distance which are both workers and soldiers

\* Expected - returns true

\*

\* Test data - A single ant within viewing distance which is not a soldier

\* Expected - returns false

\*

\* Test data - A single ant within viewing distance which is a soldier

\* Expected - returns true

\*/

var SoldiersInViewTest = new testCase('Test that soldier in view returns true when there are soldiers in view');

SoldiersInViewTest.autorun = true;

var testAntSpecies = {chars : {eyesight : 10, eyeAngle : Math.PI\*2}}; // simulate a species

var testAnt = new Soldier(1, {x : 25, y : 25});

testAnt.species = testAntSpecies;

testAnt.direction = 0;

SoldiersInViewTest.callwith = testAnt;

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

// 1

createMap();

testAnt.scan();

SoldiersInViewTest.createTest(testAnt.soldiersInView, [], 'equal', false);

// 2

createMap();

// Simulate the dummy ants

MAP[coordToIndex({x : 26, y : 24})].ant = [{id : 1, species : testAntSpecies, type : ANT\_TYPE.soldier}];

MAP[coordToIndex({x : 24, y : 22})].ant = [{id : 2, species : testAntSpecies, type : ANT\_TYPE.worker}];

MAP[coordToIndex({x : 27, y : 18})].ant = [{id : 3, species : testAntSpecies, type : ANT\_TYPE.nest}];

testAnt.scan();

SoldiersInViewTest.createTest(testAnt.soldiersInView, [], 'equal', true);

// 3

createMap();

// Simulate the dummy ants

MAP[coordToIndex({x : 24, y : 22})].ant = [{id : 1, species : testAntSpecies, type : ANT\_TYPE.worker}];

testAnt.scan();

SoldiersInViewTest.createTest(testAnt.soldiersInView, [], 'equal', false);

// 4

createMap();

// Simulate the dummy ants

MAP[coordToIndex({x : 23, y : 24})].ant = [{id : 1, species : testAntSpecies, type : ANT\_TYPE.soldier}];

testAnt.scan();

SoldiersInViewTest.createTest(testAnt.soldiersInView, [], 'equal', true);

SoldiersInViewTest.summery();

/\*\*

\* Test case - Test that soldier in view correctly determines if there are friendly (i.e. of same species) ants in view

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\*

\* Test data - A solider ant of the same species

\* Expected - returns true

\*

\* Test data - A soldier ant of a different species

\* Expected - returns false

\*/

var SoldiersInViewOfSameSpeciesTest = new testCase('Test that soldier in view correctly determines if there are friendly (i.e. of same species) ants in view');

SoldiersInViewOfSameSpeciesTest.autorun = true;

var testAntSpecies1 = {chars : {eyesight : 10, eyeAngle : Math.PI\*2}}; // simulate a species

var testAntSpecies2 = {chars : {eyesight : 11, eyeAngle : Math.PI\*2}}; // simulate a species

var testAnt = new Soldier(1, {x : 25, y : 25});

testAnt.species = testAntSpecies1;

testAnt.direction = 0;

SoldiersInViewOfSameSpeciesTest.callwith = testAnt;

MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

CELL\_SIZE = {width : 10, height : 10};

GRID\_SIZE = {width : 50, height : 50};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

// 1

createMap();

MAP[coordToIndex({x : 26, y : 24})].ant = [{id : 1, species : testAntSpecies1, type : ANT\_TYPE.soldier}];

testAnt.scan();

SoldiersInViewOfSameSpeciesTest.createTest(testAnt.soldiersInView, [], 'equal', true);

// 2

createMap();

MAP[coordToIndex({x : 26, y : 24})].ant = [{id : 1, species : testAntSpecies2, type : ANT\_TYPE.soldier}];

testAnt.scan();

SoldiersInViewOfSameSpeciesTest.createTest(testAnt.soldiersInView, [], 'equal', false);

SoldiersInViewOfSameSpeciesTest.summery();

## angleTo.test

// ---------- angleTo ----------

function arrowTo(coord1, coord2, angle) {

window.onload = function () {

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

drawRect(ctx, scaleCoord(coord1), CELL\_SIZE, '#FF0000');

drawRect(ctx, scaleCoord(coord2), CELL\_SIZE, '#0000FF');

ctx.save();

var scaledCoord = scaleCoord(coord1);

// Translate and rotate the canvas (done so can draw at an angle)

ctx.translate(scaledCoord.x + CELL\_SIZE.width/2, scaledCoord.y + CELL\_SIZE.height/2);

ctx.rotate(angle);

drawLine(ctx, {x : 0, y : 0}, {x : 0, y : -100}, '#FF0000', 1);

drawLine(ctx, {x : 0, y : -100}, {x : 5, y : -95}, '#FF0000', 1);

drawLine(ctx, {x : 0, y : -100}, {x : -5, y : -95}, '#FF0000', 1);

ctx.restore();

}

}

/\*\*

\* Test case - Test angleTo returns the correct angle to get from coord to target

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\*

\* Test data - coord : {x : 5, y : 5}, target : {x : 7, y : 5} [functional value]

\* Expected - Math.PI/2 ~= 1.5707

\*

\* Test data - coord : {x : 5, y : 5}, target : {x : 7, y : 7} [functional value]

\* Expected - 3 \* Math.PI/4 ~= 2.35619

\*

\* Test data - coord : {x : 5, y : 5}, target : {x : 5, y : 8} [functional value]

\* Expected - Math.PI ~= 3.1415

\*

\* Test data - coord : {x : 5, y : 5}, target : {x : 3, y : 4} [functional value]

\* Expected - ~= -1.10714

\*

\* Test data - coord : {x : 5, y : 5}, target : {x : 5, y : 5} [edge case]

\* Expected - Math.PI/2 ~= 1.5707

\*

\* Test data - coord : {x : 8, y : 8}, target : {x : 1, y : 1} [edge case]

\* Expected - 3 \* Math.PI/4 ~= 2.35619

\*

\* Test data - coord : {x : 7, y : 8}, target : {x : 3, y : 2} [edge case]

\* Expected - ~= 3.38657

\*/

var MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

var CELL\_SIZE = {width : 50, height : 50};

var GRID\_SIZE = {width : 10, height : 10};

var angleToTest = new testCase('Test angleTo returns the correct angle to get from coord to target');

angleToTest.callback = arrowTo;

angleToTest.callbackArgs = [{x : 5, y : 5}, {x : 7, y : 5}];

angleToTest.createTest(angleTo, [{x : 5, y : 5}, {x : 7, y : 5}], 'equal', Math.PI/2);

angleToTest.callbackArgs = [{x : 5, y : 5}, {x : 7, y : 7}];

angleToTest.createTest(angleTo, [{x : 5, y : 5}, {x : 7, y : 7}], 'equal', 3 \* Math.PI/4);

angleToTest.callbackArgs = [{x : 5, y : 5}, {x : 5, y : 8}];

angleToTest.createTest(angleTo, [{x : 5, y : 5}, {x : 5, y : 8}], 'equal', Math.PI);

angleToTest.callbackArgs = [{x : 5, y : 5}, {x : 3, y : 4}];

angleToTest.createTest(angleTo, [{x : 5, y : 5}, {x : 3, y : 4}], 'approx', -1.10714);

angleToTest.callbackArgs = [{x : 5, y : 5}, {x : 5, y : 5}];

angleToTest.createTest(angleTo, [{x : 5, y : 5}, {x : 5, y : 5}], 'equal', Math.PI/2);

angleToTest.callbackArgs = [{x : 8, y : 8}, {x : 1, y : 1}];

angleToTest.createTest(angleTo, [{x : 8, y : 8}, {x : 1, y : 1}], 'equal', 3 \* Math.PI/4);

angleToTest.callbackArgs = [{x : 7, y : 8}, {x : 3, y : 2}];

angleToTest.createTest(angleTo, [{x : 7, y : 8}, {x : 3, y : 2}], 'approx', 3.38657);

angleToTest.testAll();

angleToTest.summery();

/\* TEST FAILED

Failed

angleToTest.callbackArgs = [{x : 8, y : 8}, {x : 1, y : 1}];

angleToTest.createTest(angleTo, [{x : 8, y : 8}, {x : 1, y : 1}], 'equal', 3 \* Math.PI/4);

Old function :

function angleTo(coord, target) {

var dx = target.x - coord.x;

var dy = target.y - coord.y;

return Math.atan2(dy, dx) + Math.PI/2;

}

New function :

function angleTo(coord, target) {

if (GRID\_SIZE.width - Math.abs(target.x - coord.x) > Math.abs(target.x - coord.x)) {

dx = target.x - coord.x;

} else {

dx = GRID\_SIZE.width - (target.x - coord.x);

}

if (GRID\_SIZE.height - Math.abs(target.y - coord.y) > Math.abs(target.y - coord.y)) {

dy = target.y - coord.y;

} else {

dy = GRID\_SIZE.height - (target.y - coord.y);

}

return Math.atan2(dy, dx) + Math.PI/2;

}

\*/

## boundary.test

// ---------- boundary ----------

/\*\*

\* Test case - Test that boundary returns the correct warped coordinate

\*

\* Environment:

\* bounds = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\*

\* Test data - {x : 4, y : 2} [functional value]

\* Expected - {x : 4, y : 2}

\*

\* Test data - {x : 10, y : 10} [edge case]

\* Expected - {x : 0, y : 0}

\*

\* Test data - {x : 10, y : 4} [edge case]

\* Expected - {x : 0, y : 4}

\*

\* Test data - {x : 7, y : 10} [edge case]

\* Expected - {x : 7, y : 0}

\*/

var bounds = {x : {min : 0, max : 10}, y : {min : 0, max : 10}};

var CELL\_SIZE = {width : 50, height : 50};

var GRID\_SIZE = {width : 10, height : 10};

function boundaryTest(ctx, coord) {

drawRect(ctx, scaleCoord(coord), CELL\_SIZE, '#FF0000');

drawRect(ctx, scaleCoord(boundary(coord, bounds)), CELL\_SIZE, '#0000FF');

log(' For ' + format(coord) + ' boundary is ' + format(boundary(coord, bounds)));

}

window.onload = function () {

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

log('Test that boundary returns the correct warped coordinate');

boundaryTest(ctx, {x : 4, y : 2});

boundaryTest(ctx, {x : 10, y : 10});

boundaryTest(ctx, {x : 10, y : 4});

boundaryTest(ctx, {x : 7, y : 10});

};

## clone.test

// ---------- clone ----------

/\*\*

\* Test case - Test clone produces an exact clone of an object

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\*

\* Test data - {a : [1, 2, 3], b : {c : 4, d : '5'}, e : {f : {g : 6}}, h : [['end', 'of', ['object']]]}

\* Expected - {a : [1, 2, 3], b : {c : 4, d : '5'}, e : {f : {g : 6}}, h : [['end', 'of', ['object']]]}

\*/

var cloneExactCloneTest = new testCase('Test clone produces an exact clone of an object');

cloneExactCloneTest.createTest(clone, [{a : [1, 2, 3], b : {c : 4, d : '5'}, e : {f : {g : 6}}, h : [['end', 'of', ['object']]]}], 'equal', {a : [1, 2, 3], b : {c : 4, d : '5'}, e : {f : {g : 6}}, h : [['end', 'of', ['object']]]});

cloneExactCloneTest.testAll();

cloneExactCloneTest.summery();

/\*\*

\* Test case - Test clone produces a copy of the object i.e. not by reference

\* as:

\* var a = {prop1 : 'spam', prop2 : 'eggs'};

\* var b = {prop1 : 'spam', prop2 : 'eggs'};

\* var c = a; // by reference

\*

\* a === b -> FALSE

\* a === c -> TRUE

\* b === c -> FALSE

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\*

\* Test data - let b = {a : 1}

\* Expected - {a : 1} != b

\*/

var cloneCopyTest = new testCase('Test clone produces a copy of the object i.e. not by reference');

cloneCopyTest.evaluateTo = false;

cloneCopyTest.createTest(clone, [{a : 1}], 'exactlyEqual', {a : 1});

cloneCopyTest.testAll();

cloneCopyTest.summery();

## coordToIndex.test

// ---------- coordToIndex ----------

/\*\*

\* Test case - Test coordToIndex returns the correct index for a specific coordinate

\* Environment- GRID\_SIZE = {10, 10}

\*

\* Test data - {x : 0, y : 0} [edge case]

\* Expected - 0

\*

\* Test data - {x : 5, y : 2} [functional value]

\* Expected - 25

\*/

var coordToIndexTest = new testCase('Test coordToIndex returns the correct index for a specific coordinate');

GRID\_SIZE = {width : 10, height : 10};

coordToIndexTest.createTest(coordToIndex, [{x : 0, y : 0}], 'equal', 0);

coordToIndexTest.createTest(coordToIndex, [{x : 5, y : 2}], 'equal', 25);

coordToIndexTest.testAll();

coordToIndexTest.summery();

## distance.test

// ---------- distance ----------

/\*\*

\* Test case - Check it returns the correct distance

\*

\*

\* GRID\_SIZE = {width : 10, height : 10}

\*

\* Test data - coord1 : {x : 0, y : 0}, coord2 : {x : 0, y : 0} [edge case]

\* Expected - 0

\*

\* Test data - coord1 : {x : -2, y : 1}, coord2 : {x : 1, y : 5} [functional value]

\* Expected - 5

\*/

var distanceTest = new testCase('Check it returns the correct distance');

distanceTest.createTest(distance, [{x : 0, y : 0}, {x : 0, y : 0}], 'equal', 0);

distanceTest.createTest(distance, [{x : -2, y : 1}, {x : 1, y : 5}], 'equal', 5);

distanceTest.testAll();

distanceTest.summery();

## getBlock.test

// ---------- getBlock ----------

function drawBlocks(blocks) {

window.onload = function () {

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

for (var i = 0; i < blocks.length; i++) {

drawRect(ctx, scaleCoord(blocks[i]), CELL\_SIZE, '#0000FF');

}

}

}

/\*\*

\* Test case - Test getBlock returns a block of cells the correct size

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\*

\* Test data - coord : {x : 5, y : 5}, size : {width : 0, height : 0} [edge case]

\* Expected - A single block centred at {x : 5, y : 5}

\*

\* Test data - coord : {x : 5, y : 5}, size : {width : 3, height : 3} [functional value]

\* Expected - A 7 wide and 7 tall block centred at {x : 5, y : 5}

\*

\* Test data - coord : {x : 5, y : 5}, size : {width : 1, height : 3} [functional value]

\* Expected - A 3 wide and 7 tall block centred at {x : 5, y : 5}

\*/

var MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

var CELL\_SIZE = {width : 50, height : 50};

var GRID\_SIZE = {width : 10, height : 10};

var getBlockSizeTest = new testCase('Test getBlock returns a block of cells the correct size');

getBlockSizeTest.callback = drawBlocks;

getBlockSizeTest.createTest(getBlock, [{x : 5, y : 5}, {width : 0, height : 0}], 'desc', 'A single block centred at {x : 5, y : 5}');

getBlockSizeTest.createTest(getBlock, [{x : 5, y : 5}, {width : 3, height : 3}], 'none', 'A 7 wide and 7 tall block centred at {x : 5, y : 5}');

getBlockSizeTest.createTest(getBlock, [{x : 5, y : 5}, {width : 1, height : 3}], 'none', 'A 3 wide and 7 tall block centred at {x : 5, y : 5}');

getBlockSizeTest.testAll();

/\*\*

\* Test case - Test getBlock returns a block of cells the correct position

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\*

\* Test data - coord : {x : 1, y : 3}, size : {width : 2, height : 2} [edge case]

\* Expected - A 5 wide and 5 tall block centred at {x : 1, y : 3} which wraps to the other side of the map

\*

\* Test data - coord : {x : 0, y : 0}, size : {width : 1, height : 1} [edge case]

\* Expected - A 3 wide and 3 tall block centred at {x : 0, y : 0} which wraps to every corner of the map

\*

\*/

var MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

var CELL\_SIZE = {width : 50, height : 50};

var GRID\_SIZE = {width : 10, height : 10};

var getBlockPositionTest = new testCase('Test getBlock returns a block of cells the correct position');

getBlockPositionTest.callback = drawBlocks;

getBlockPositionTest.createTest(getBlock, [{x : 1, y : 3}, {width : 2, height : 2}], 'none', 'A 5 wide and 5 tall block centred at {x : 1, y : 3} which wraps to the other side of the map');

getBlockPositionTest.createTest(getBlock, [{x : 0, y : 0}, {width : 1, height : 1}], 'none', 'A 3 wide and 3 tall block centred at {x : 0, y : 0} which wraps to every corner of the map');

getBlockPositionTest.testAll();

## getCellCoord.test

// ---------- getCellCoord ----------

/\*\*

\* Test case - Test that getCellCoord returns the correct cell for a particular coordinate

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\*

\* Test data - {x : 7.2, y : 3.6} [functional value]

\* Expected - {x : 7, y : 3}

\*

\*/

var MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}};

var CELL\_SIZE = {width : 50, height : 50};

var GRID\_SIZE = {width : 10, height : 10};

function getCellCoordTest(ctx, coord) {

drawArc(ctx, scaleCoord(coord), 5, 0, 2 \* Math.PI, '#000000', 1, '#FF0000');

drawRect(ctx, scaleCoord(getCellCoord(coord)), CELL\_SIZE, '#0000FF');

log(' For ' + format(coord) + ' getCellCoord is ' + format(getCellCoord(coord)));

}

window.onload = function () {

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

log('Test that getCellCoord returns the correct cell for a particular coordinate');

getCellCoordTest(ctx, {x : 7.2, y : 3.6});

};

// FAIL

/\*

getCellCoordTest(ctx, {x : 7.2, y : 3.5}) test failed.

Original function:

function getCellCoord(coord) {

return boundary({x: Math.round(coord.x), y: Math.round(coord.y)}, MAP\_BOUNDARY);

}

Fixed function:

function getCellCoord(coord) {

return boundary({x: Math.floor(coord.x), y: Math.floor(coord.y)}, MAP\_BOUNDARY);

}

\*/

## getSector.test

// ---------- getSector ----------

function drawBlocks(centerBlock, blocks) {

window.onload = function () {

var ctx = getElement(CANVAS.name).getContext('2d');

drawBackground(ctx);

simpleGrid(ctx);

for (var i = 0; i < blocks.length; i++) {

drawRect(ctx, scaleCoord(blocks[i]), CELL\_SIZE, '#0000FF');

}

drawRect(ctx, scaleCoord(centerBlock), CELL\_SIZE, '#FF0000');

}

}

/\*\*

\* Test case - Test getSector returns sector of correct radius

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\*

\* Test data - coord : {x : 25, y : 25}, radius : 15, direction : 0, angle : 2 \* Math.PI [functional value]

\* Expected - A circle of radius 15 centred at {x : 25, y : 25}

\*

\* Test data - coord : {x : 25, y : 25}, radius : 6, direction : 0, angle : 2 \* Math.PI [functional value]

\* Expected - A circle of radius 6 centred at {x : 25, y : 25}

\*/

var MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

var CELL\_SIZE = {width : 10, height : 10};

var GRID\_SIZE = {width : 50, height : 50};

var getSectorRadiusTest = new testCase('Test getSector returns sector of correct radius');

getSectorRadiusTest.callback = drawBlocks;

getSectorRadiusTest.callbackArgs = [{x : 25, y : 25}];

getSectorRadiusTest.createTest(getSector, [{x : 25, y : 25}, 15, 0, 2 \* Math.PI], 'desc', 'A circle of radius 15 centred at {x : 25, y : 25}');

getSectorRadiusTest.createTest(getSector, [{x : 25, y : 25}, 6, 0, 2 \* Math.PI], 'desc', 'A circle of radius 6 centred at {x : 25, y : 25}');

getSectorRadiusTest.testAll();

getSectorRadiusTest.summery();

/\*\*

\* Test case - Test getSector returns sector at the correct angle

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\*

\* Test data - coord : {x : 25, y : 25}, radius : 15, direction : 0, angle : Math.PI [functional value]

\* Expected - A sector of radius 15 centred at {x : 25, y : 25} with an angle of Math.PI i.e. a semi circle

\*

\* Test data - coord : {x : 25, y : 25}, radius : 15, direction : 0, angle : Math.PI/4 [functional value]

\* Expected - A sector of radius 15 centred at {x : 25, y : 25} with an angle of Math.PI/4

\*/

var MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

var CELL\_SIZE = {width : 10, height : 10};

var GRID\_SIZE = {width : 50, height : 50};

var getSectorAngleTest = new testCase('Test getSector returns sector at the correct angle');

getSectorAngleTest.callback = drawBlocks;

getSectorAngleTest.callbackArgs = [{x : 25, y : 25}];

getSectorAngleTest.createTest(getSector, [{x : 25, y : 25}, 15, 0, Math.PI], 'desc', 'A sector of radius 15 centred at {x : 25, y : 25} with an angle of Math.PI i.e. a semi circle');

getSectorAngleTest.createTest(getSector, [{x : 25, y : 25}, 15, 0, Math.PI / 4], 'desc', 'A sector of radius 15 centred at {x : 25, y : 25} with an angle of Math.PI/4');

getSectorAngleTest.testAll();

getSectorAngleTest.summery();

/\*\*

\* Test case - Test getSector returns sector in correct direction

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\*

\* Test data - coord : {x : 25, y : 25}, radius : 15, direction : Math.PI, angle : Math.PI/4 [functional value]

\* Expected - A sector of radius 15 centred at {x : 25, y : 25} with an angle of Math.PI/4 pointing downwards

\*

\* Test data - coord : {x : 25, y : 25}, radius : 15, direction : 6 \* Math.PI / 4, angle : Math.PI/4 [functional value]

\* Expected - A sector of radius 15 centred at {x : 25, y : 25} with an angle of Math.PI/4 pointing along the centre line of the 4th and 3rd quadrants

\*

\* Test data - coord : {x : 25, y : 25}, radius : 15, direction : 3.657, angle : Math.PI/4 [functional value]

\* Expected - A sector of radius 15 centred at {x : 25, y : 25} with an angle of Math.PI/4 pointing along the centre line of 3.657 radians

\*/

var MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

var CELL\_SIZE = {width : 10, height : 10};

var GRID\_SIZE = {width : 50, height : 50};

var getSectorDirectionTest = new testCase('Test getSector returns sector in correct direction');

getSectorDirectionTest.callback = drawBlocks;

getSectorDirectionTest.callbackArgs = [{x : 25, y : 25}];

getSectorDirectionTest.createTest(getSector, [{x : 25, y : 25}, 15, Math.PI, Math.PI/4], 'desc', 'A sector of radius 15 centred at {x : 25, y : 25} with an angle of Math.PI/4 pointing downwards');

getSectorDirectionTest.createTest(getSector, [{x : 25, y : 25}, 15, 6 \* Math.PI / 4, Math.PI/4], 'desc', 'A sector of radius 15 centred at {x : 25, y : 25} with an angle of Math.PI/4 pointing along the centre line of the 4th and 3rd quadrants');

getSectorDirectionTest.createTest(getSector, [{x : 25, y : 25}, 15, 3.657, Math.PI/4], 'desc', 'A sector of radius 15 centred at {x : 25, y : 25} with an angle of Math.PI/4 pointing along the center line of 3.657 radians');

getSectorDirectionTest.testAll();

getSectorDirectionTest.summery();

## indexToCoord.test

// ---------- indexToCoord ----------

/\*\*

\* Test case - Test that indexToCoord returns the correct coordinate from a specific index

\*

\* Test data - 25 [functional value]

\* Expected - {x : 5, y : 2}

\*

\* Test data - 0 [edge case]

\* Expected - {x : 0, y : 0}

\*/

var indexToCoordTest = new testCase('Test that indexToCoord returns the correct coordinate from a specific index');

GRID\_SIZE = {width : 10, height : 10};

indexToCoordTest.createTest(indexToCoord, [25], 'equal', {x : 5, y : 2});

indexToCoordTest.createTest(indexToCoord, [0], 'equal', {x : 0, y : 0});

indexToCoordTest.testAll();

indexToCoordTest.summery();

## randColour.test

// ---------- randColour ----------

/\*\*

\* Test case - Test randColour generates random colours (3 tests)

\* Test data - N/A

\* Expected - To see random colours being generated

\*/

var randColourTest = new testCase('Test randColour generates random colours');

randColourTest.createTest(randColour, [], 'desc', 'A HEX colour');

randColourTest.createTest(randColour, [], 'desc', 'A HEX colour');

randColourTest.createTest(randColour, [], 'desc', 'A HEX colour');

randColourTest.testAll();

randColourTest.summery();

## randFloat.test

// ---------- randFloat ----------

/\*\*

\* Test case - Test randFloat returns a value within a specific range

\* Test data - {min : 0, max : 2}

\* Expected - 0 <= value <= 2

\* Test data - {min : 0, max : 0}

\* Expected - 0 <= value <= 0

\* Test data - {min : -2, max : 0}

\* Expected - -2 <= value <= 0

\* Test data - {min : 1, max : 2}

\* Expected - 1 <= value <= 2

\* Test data - {min : -1, max : 1}

\* Expected - -1 <= value <= 1

\* Test data - {min : 0, max : 0.1}

\* Expected - 0 <= value <= 0.1

\* Test data - {min : -0.1, max : 0.1}

\* Expected - -0.1 <= value <= 0.1

\*/

var randFloatTest = new testCase('Test randFloat returns a value within a specific range');

randFloatTest.createTest(randFloat, [{min : 0, max : 2}], 'range', {min : 0, max : 2});

randFloatTest.createTest(randFloat, [{min : 0, max : 0}], 'range', {min : 0, max : 0});

randFloatTest.createTest(randFloat, [{min : -2, max : 0}], 'range', {min : -2, max : 0});

randFloatTest.createTest(randFloat, [{min : 1, max : 2}], 'range', {min : 1, max : 2});

randFloatTest.createTest(randFloat, [{min : -1, max : 1}], 'range', {min : -1, max : 1});

randFloatTest.createTest(randFloat, [{min : 0, max : 0.1}], 'range', {min : 0, max : 0.1});

randFloatTest.createTest(randFloat, [{min : -0.1, max : 0.1}], 'range', {min : -0.1, max : 0.1});

var numOfRuns = 100; // The number of runs of the function to test the range

for (var i = 0; i < numOfRuns; i++) {

randFloatTest.testAll();

}

randFloatTest.summery();

## randInt.test

// ---------- randInt ----------

/\*\*

\* Test case - Test randInt returns a value within a specific range

\* Test data - {min : 0, max : 100}

\* Expected - 0 <= value <= 100

\* Test data - {min : 0, max : 0}

\* Expected - 0 <= value <= 0

\* Test data - {min : -100, max : 0}

\* Expected - -100 <= value <= 0

\* Test data - {min : 20, max : 40}

\* Expected - 20 <= value <= 40

\* Test data - {min : -20, max : 20}

\* Expected - -20 <= value <= 20

\* Test data - {min : 0, max : 1}

\* Expected - 0 <= value <= 1

\*/

var randIntTest = new testCase('Test randInt returns a value within a specific range');

randIntTest.createTest(randInt, [{min : 0, max : 100}], 'range', {min : 0, max : 100});

randIntTest.createTest(randInt, [{min : 0, max : 0}], 'range', {min : 0, max : 0});

randIntTest.createTest(randInt, [{min : -100, max : 0}], 'range', {min : -100, max : 0});

randIntTest.createTest(randInt, [{min : 20, max : 40}], 'range', {min : 20, max : 40});

randIntTest.createTest(randInt, [{min : -20, max : 20}], 'range', {min : -20, max : 20});

randIntTest.createTest(randInt, [{min : 0, max : 1}], 'range', {min : 0, max : 1});

var numOfRuns = 100; // The number of runs of the function to test the range

for (var i = 0; i < numOfRuns; i++) {

randIntTest.testAll();

}

randIntTest.summery();

## randProperty.test

// ---------- randProperty ----------

/\*\*

\* Test case - Test randProperty returns a random property from an object literal

\*

\* Test data - {a : 0, b : 1, c : 2} [edge case] (run three times)

\* Expected - either 'a', 'b' or 'c'

\*/

var randPropertyTest = new testCase('Test randProperty returns a random property from an object literal');

randPropertyTest.createTest(randProperty, [{a : 0, b : 1, c : 2}], 'desc', 'A random property from an object literal');

randPropertyTest.createTest(randProperty, [{a : 0, b : 1, c : 2}], 'desc', 'A random property from an object literal');

randPropertyTest.createTest(randProperty, [{a : 0, b : 1, c : 2}], 'desc', 'A random property from an object literal');

randPropertyTest.createTest(randProperty, [{a : 0, b : 1, c : 2}], 'desc', 'A random property from an object literal');

randPropertyTest.createTest(randProperty, [{a : 0, b : 1, c : 2}], 'desc', 'A random property from an object literal');

randPropertyTest.testAll();

randPropertyTest.summery();

## scaleCoord.test

// ---------- scaleCoord ----------

/\*\*

\* Test case - Test that scaleCoord scales coordinates correctly

\*

\* Environment:

\* CELL\_SIZE = {width : 5, height : 5};

\* CANVAS\_OFFSET = {x : 3, y : 2};

\*

\* Test data - {x : 4, y : 2} [functional value]

\* Expected - {x : 23, y : 12}

\*/

var scaleCoordTest = new testCase('Test that scaleCoord scales coordinates correctly');

CELL\_SIZE = {width : 5, height : 5};

CANVAS\_OFFSET = {x : 3, y : 2};

scaleCoordTest.createTest(scaleCoord, [{x : 4, y : 2}], 'equal', {x : 23, y : 12});

scaleCoordTest.testAll();

scaleCoordTest.summery();

## validateDirection.test

// ---------- validateDirection ----------

/\*\*

\* Test case - Test randInt returns the correct value

\* Test data - 5 \* Math.PI [Functioning value]

\* Expected - 1 \* Math.PI

\* Test data - 2 \* Math.PI [edge case]

\* Expected - 0

\* Test data - 0 [edge case]

\* Expected - 0

\* Test data - 4 [[Functioning value] non multiple of PI]

\* Expected - 4

\* Test data - -2 \* Math.PI [negitive [Functioning value]]

\* Expected - 0

\*/

var validateDirectionTest = new testCase('Test randInt returns the correct value');

validateDirectionTest.createTest(validateDirection, [5 \* Math.PI], 'equal', 1 \* Math.PI);

validateDirectionTest.createTest(validateDirection, [1 \* Math.PI], 'equal', 1 \* Math.PI);

validateDirectionTest.createTest(validateDirection, [0], 'equal', 0);

validateDirectionTest.createTest(validateDirection, [4], 'equal', 4);

validateDirectionTest.createTest(validateDirection, [-2 \* Math.PI], 'equal', 0);

validateDirectionTest.testAll();

validateDirectionTest.summery();

## Worker.depositingFood.test

// ---------- Worker depositing Food ----------

function tick() {

testAnt.update();

simpleGrid(CTX);

for (var i = 0; i < NUM\_OF\_CELLS; i++) {

if (MAP[i].ant.length > 0) {

MAP[i].ant[0].draw(CTX);

} else if (MAP[i].pheromone.length > 0) {

var pheromone = MAP[i].pheromone[0];

pheromone.draw(CTX);

}

}

if (testAnt.carrying === 0)

console.log('Food all deposited.');

if (testAnt.goal === GOAL.dropFood) {

setTimeout(tick, 1000);

console.log('searching for nest');

} else if (testAnt.goal === GOAL.findFood) {

console.log('Goal updated to GOAl.findFood');

}

}

/\*\*

\* Test case - Simulate an ant dropping food off at the nest

\*

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* A single ant and nest

\*/

var WorkerAntDepositingFoodTest = new testCase('Simulate an ant dropping food off at the nest');

WorkerAntDepositingFoodTest.autorun = true;

// Configure environment

MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

CELL\_SIZE = {width : 50, height : 50};

GRID\_SIZE = {width : 10, height : 10};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

STARTING\_QUEEN\_ANT\_NUMBER = 0;

var CTX;

// Create the ant

var testAnt = new Worker(genID(), {x : 5, y : 5});

testAnt.species = new Species(); // starts with default values

testAnt.size = CELL\_SIZE; // starts with default values

testAnt.direction = 0;

testAnt.carrying = 3;

testAnt.goal = GOAL.dropFood;

createMap();

// Add Nest

var nest = new Nest(genID(), {x : 2, y : 2});

nest.species = testAnt.species;

testAnt.nest = nest;

nest.createNest();

WorkerAntDepositingFoodTest.createTest('N/A', [], 'desc', 'watch ant searching dropping off food at the nest.');

window.onload = function () {

CTX = getElement(CANVAS.name).getContext('2d');

setTimeout(tick, 1000);

}

WorkerAntDepositingFoodTest.summery();

## Worker.draw.test

// ---------- Worker.draw ----------

/\*\*

\* Test case - Test draw draws the ant correctly

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 50}, y : {min : 0, max : 50}}

\* CELL\_SIZE = {width : 10, height : 10};

\* GRID\_SIZE = {width : 50, height : 50};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* MAX\_PHEROMONE\_CONCENTRATION = 1;

\* A single ant in a static position with pheromones around it

\*

\* Test data - Ant carrying food

\* Expected - Black square with white square in the centre

\*

\* Test data - Ant not carrying food

\* Expected - Black square

\*/

window.onload = function() {

var AntDrawTest = new testCase('Test draw draws the ant correctly');

AntDrawTest.autorun = true;

CANVAS\_OFFSET = {x : 0, y : 0};

MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

CELL\_SIZE = {width : 50, height : 50};

GRID\_SIZE = {width : 10, height : 10};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

var ctx = getElement(CANVAS.name).getContext('2d');

simpleGrid(ctx);

var testAnt = new Worker(1, {x : 5, y : 5});

testAnt.direction = 4.104;

testAnt.carrying = 1;

testAnt.size = CELL\_SIZE;

AntDrawTest.callwith = testAnt;

AntDrawTest.createTest(testAnt.draw, [ctx], 'desc', 'Black square with white square in the centre');

// 2

testAnt.carrying = 0;

AntDrawTest.createTest(testAnt.draw, [ctx], 'desc', 'Black square');

AntDrawTest.summery();

}

## Worker.findingFood.test

// ---------- Worker finding Food ----------

function basicMapDraw() {

console.log('b')

var ctx = getElement(CANVAS.name).getContext('2d');

for (var i = 0; i < NUM\_OF\_CELLS; i++) {

if (MAP[i].ant.length > 0) {

MAP[i].ant[0].draw(ctx);

}

else if (MAP[i].food !== void(0))

MAP[i].food.draw(ctx);

}

}

function tick() {

testAnt.update();

simpleGrid(CTX);

testAnt.draw(CTX);

MAP[coordToIndex({x : 2, y : 8})].food.draw(CTX);

var blocks = getSector(testAnt.coord, testAnt.species.chars.eyesight,

testAnt.direction, testAnt.species.chars.eyeAngle)

for (var i = 0; i < blocks.length; i++) {

drawRect(CTX, scaleCoord(blocks[i]), CELL\_SIZE, '#0000FF');

}

if (testAnt.target !== void(0))

console.log('Found food target.')

if (testAnt.goal === GOAL.findFood) {

setTimeout(tick, 1000);

console.log('searching for food');

} else if (testAnt.goal === GOAL.getFood) {

console.log('Goal updated to GOAl.getFood');

}

}

/\*\*

\* Test case - Simulate a single ant looking for food

\* GOAL -> findFood,

\* wonder + scan + findFoodTarget,

\* if spot food:

\* GOAL.getFood

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* A single ant in a static position with multiple ants around it

\*

\* Test data - A single ant and a single piece of food

\* Expected - this.itemsInView.ants.length = 0 i.e. cannot see any ants in view

\*/

var WorkerAntFindingFoodTest = new testCase('Simulate a single worker ant looking for food');

WorkerAntFindingFoodTest.autorun = true;

// Configure environment

MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

CELL\_SIZE = {width : 50, height : 50};

GRID\_SIZE = {width : 10, height : 10};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

STARTING\_QUEEN\_ANT\_NUMBER = 0;

var CTX;

// Create the ant

var testAnt = new Worker(genID(), {x : 5, y : 5});

testAnt.species = new Species(); // starts with default values

testAnt.size = CELL\_SIZE; // starts with default values

createMap();

// Add food

var fs = new FoodSystem();

fs.variation.max = 1;

MAP[coordToIndex({x : 2, y : 8})].food = new Food(fs, 1, {x : 2, y : 8});

WorkerAntFindingFoodTest.createTest('N/A', [], 'desc', 'watch ant searching for food, targeting food and updating its goal.');

window.onload = function () {

CTX = getElement(CANVAS.name).getContext('2d');

setTimeout(tick, 1000);

}

WorkerAntFindingFoodTest.summery();

## Worker.gettingFood.test

// ---------- Worker getting Food ----------

function basicMapDraw() {

console.log('b')

var ctx = getElement(CANVAS.name).getContext('2d');

for (var i = 0; i < NUM\_OF\_CELLS; i++) {

if (MAP[i].ant.length > 0) {

MAP[i].ant[0].draw(ctx);

}

else if (MAP[i].food !== void(0))

MAP[i].food.draw(ctx);

}

}

function tick() {

testAnt.update();

simpleGrid(CTX);

testAnt.draw(CTX);

MAP[coordToIndex({x : 5, y : 2})].food.draw(CTX);

var blocks = getSector(testAnt.coord, testAnt.species.chars.eyesight,

testAnt.direction, testAnt.species.chars.eyeAngle)

for (var i = 0; i < blocks.length; i++) {

drawRect(CTX, scaleCoord(blocks[i]), CELL\_SIZE, '#0000FF');

}

if (testAnt.target !== void(0))

console.log('Found food target.')

if (testAnt.goal === GOAL.findFood) {

setTimeout(tick, 1000);

console.log('searching for food');

} else if (testAnt.goal === GOAL.getFood) {

console.log('Goal updated to GOAl.getFood');

}

}

/\*\*

\* Test case - Simulate a single ant retrieving food

\* GOAL -> getFood,

\* getFood()

\* if at food:

\* useFood()

\* while food still left:

\* if hungry():

\* takeFood() (health +1 \* FOOD\_HEALTH\_RATIO, food amount -1)

\* if canCarry()

\* takeFood() (carrying +1, food amount -1)

\* set food to void(0), target to void(0)

\* GOAL -> deposit food

\*

\*

\* Environment:

\* MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

\* CELL\_SIZE = {width : 50, height : 50};

\* GRID\_SIZE = {width : 10, height : 10};

\* NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

\* A single ant near a single piece of food

\*/

var WorkerAntGettingFoodTest = new testCase('Simulate a single ant retrieving food');

WorkerAntGettingFoodTest.autorun = true;

// Configure environment

MAP\_BOUNDARY = {x : {min : 0, max : 10}, y : {min : 0, max : 10}}

CELL\_SIZE = {width : 50, height : 50};

GRID\_SIZE = {width : 10, height : 10};

NUM\_OF\_CELLS = GRID\_SIZE.width \* GRID\_SIZE.height;

STARTING\_QUEEN\_ANT\_NUMBER = 0;

var CTX;

// Create the ant

var testAnt = new Worker(genID(), {x : 5, y : 5});

testAnt.species = new Species(); // starts with default values

testAnt.size = CELL\_SIZE; // starts with default values

testAnt.direction = 0;

createMap();

// Add food

var fs = new FoodSystem();

fs.variation.max = 1;

MAP[coordToIndex({x : 5, y : 2})].food = new Food(fs, 1, {x : 5, y : 2});

WorkerAntFindingFoodTest.createTest('N/A', [], 'desc', 'watch ant searching for food, targeting food and updating its goal.');

window.onload = function () {

CTX = getElement(CANVAS.name).getContext('2d');

setTimeout(tick, 1000);

}

WorkerAntFindingFoodTest.summery();

## Equal

// https://gist.github.com/stamat/5841593

//Returns the object's class, Array, Date, RegExp, Object are of interest to us

var getClass = function(val) {

return Object.prototype.toString.call(val)

.match(/^\[object\s(.\*)\]$/)[1];

};

//Defines the type of the value, extended typeof

var whatis = function(val) {

if (val === undefined)

return 'undefined';

if (val === null)

return 'null';

var type = typeof val;

if (type === 'object')

type = getClass(val).toLowerCase();

if (type === 'number') {

if (val.toString().indexOf('.') > 0)

return 'float';

else

return 'integer';

}

return type;

};

var compareObjects = function(a, b) {

if (a === b)

return true;

for (var i in a) {

if (b.hasOwnProperty(i)) {

if (!equal(a[i],b[i])) return false;

} else {

return false;

}

}

for (var i in b) {

if (!a.hasOwnProperty(i)) {

return false;

}

}

return true;

};

var compareArrays = function(a, b) {

if (a === b)

return true;

if (a.length !== b.length)

return false;

for (var i = 0; i < a.length; i++){

if(!equal(a[i], b[i])) return false;

};

return true;

};

var \_equal = {};

\_equal.array = compareArrays;

\_equal.object = compareObjects;

\_equal.date = function(a, b) {

return a.getTime() === b.getTime();

};

\_equal.regexp = function(a, b) {

return a.toString() === b.toString();

};

// uncoment to support function as string compare

// \_equal.fucntion = \_equal.regexp;

/\*

\* Are two values equal, deep compare for objects and arrays.

\* @param a {any}

\* @param b {any}

\* @return {boolean} Are equal?

\*/

var equal = function(a, b) {

if (a !== b) {

var atype = whatis(a), btype = whatis(b);

if (atype === btype)

return \_equal.hasOwnProperty(atype) ? \_equal[atype](a, b) : a==b;

return false;

}

return true;

};

## Test

/\*\*

\* A Single test to be used within a test case

\*/

var test = function(functionToTest, arguments) {

this.functionToTest = functionToTest;

this.arguments = arguments;

this.showPassMessage = false;

this.showFailMessage = true;

this.expected;

this.type; // 'equal' for an equality test or 'range' for a range check

this.callback;

this.callbackArgs = [];

this.evaluateTo = true;

this.callwith = void(0);

};

test.prototype.passMessage = function(msg) {

if (this.showPassMessage)

return 'TEST PASSED - ' + msg;

else

return '';

};

test.prototype.failMessage = function(msg) {

if (this.showFailMessage)

return 'TEST FAILED - ' + msg;

else

return '';

};

// Returns the value of the testing function run with specific args

test.prototype.run = function() {

if (typeof this.functionToTest === 'function')

var result = this.functionToTest.apply(this.callwith, this.arguments);

else

var result = this.functionToTest;

if (typeof this.callback === 'function') {

this.callbackArgs.push(result);

this.callback.apply(void(0), this.callbackArgs);

}

return result;

};

test.prototype.test = function() {

if (this.type === 'equal') {

return this.equal(this.expected);

} else if (this.type === 'exactlyEqual') {

return this.exactlyEqual(this.expected);

} else if (this.type === 'approx') {

return this.approx(this.expected);

} else if (this.type === 'range') {

return this.inRange(this.expected);

} else if (this.type === 'typeOf') {

return this.ofType(this.expected);

} else if (this.type === 'none') {

this.run();

return [void(0)];

} else if (this.type === 'desc') {

var result = this.run();

return ['desc', 'TEST DESC - Expected ' + this.expected + ', and result was ' + format(result) + '.'];

}

};

test.prototype.inRange = function(range) {

var value = this.run();

if ((value >= range.min && value <= range.max) === this.evaluateTo)

return [true, this.passMessage('Expected in range ' + format(range) + ' and was ' + format(value) + '.')];

else

return [false, this.failMessage('Expected in range ' + format(range) + ', however was ' + format(value) + '.')];

};

// Tests if the testing function returns the expected results

test.prototype.equal = function(expected) {

var value = this.run();

if (equal(value, expected) === this.evaluateTo)

return [true, this.passMessage('Expected ' + format(expected) + ' and was ' + format(value) + '.')];

else

return [false, this.failMessage('Expected ' + format(expected) + ', however was ' + format(value) + '.')];

};

// Tests if the testing function returns the expected results

test.prototype.exactlyEqual = function(expected) {

var value = this.run();

if ((value === expected) === this.evaluateTo)

return [true, this.passMessage('Expected ' + format(expected) + ' and was ' + format(value) + '.')];

else

return [false, this.failMessage('Expected ' + format(expected) + ', however was ' + format(value) + '.')];

};

// Tests if the testing function returns the expected results

test.prototype.approx = function(expected) {

var value = this.run();

var percentageError = Math.abs(Math.abs(value - expected) / expected);

if ((percentageError < 0.001) === this.evaluateTo) // accept 0.1% error

return [true, this.passMessage('Expected ' + format(expected) + ' and was ' + format(value) + '.')];

else

return [false, this.failMessage('Expected ' + format(expected) + ', however was ' + format(percentageError.toFixed(3)) + '% different.')];

};

test.prototype.ofType = function(expected) {

var value = typeof this.run();

if ((value === expected) === this.evaluateTo) {

this.passMessage('Expected ' + format(expected) + ' and was ' + format(value) + '.');

return true;

} else {

this.failMessage('Expected ' + format(expected) + ', however was ' + format(value) + '.');

return false;

}

};

## testCase

/\*\*

\* Test Case class

\*/

var testCase = function(discription) {

this.discription = discription;

this.tests = [];

this.passed = 0;

this.failed = 0;

this.numberOfTests = 0

this.callback;

this.callbackArgs = [];

this.evaluateTo = true;

this.callwith = void(0);

this.autorun = false;

this.descs = [];

this.passMessages = [];

this.failMessages = [];

};

testCase.prototype.createTest = function(functionToTest, arguments, type, expected, showPassMessage, showFailMessage) {

showPassMessage = typeof showPassMessage !== 'undefined' ? showPassMessage : true;

showFailMessage = typeof showFailMessage !== 'undefined' ? showFailMessage : true;

var singleTest = new test(functionToTest, arguments)

singleTest.showPassMessage = showPassMessage;

singleTest.showFailMessage = showFailMessage;

singleTest.type = type;

singleTest.expected = expected;

singleTest.callwith = this.callwith;

if (typeof this.callback === 'function') {

singleTest.callback = this.callback;

singleTest.callbackArgs = this.callbackArgs;

}

singleTest.evaluateTo = this.evaluateTo;

this.tests.push(singleTest);

this.numberOfTests += 1;

if (this.autorun) {

var result = singleTest.test();

this.parseResult(result);

}

return singleTest;

};

testCase.prototype.quickTest = function(functionToTest, arguments, type, expected, showPassMessage, showFailMessage) {

showPassMessage = typeof showPassMessage !== 'undefined' ? showPassMessage : false;

showFailMessage = typeof showFailMessage !== 'undefined' ? showFailMessage : true;

var test = this.createTest(functionToTest, arguments, type, expected, showPassMessage, showFailMessage, callback);

var result = test.test();

this.parseResult(result);

this.summery();

};

testCase.prototype.parseResult = function(result) {

if (result[0] === true) {

this.passed += 1;

this.passMessages.push(result[1]);

} else if (result[0] === false) {

this.failed += 1;

this.failMessages.push(result[1]);

} else if (result[0] === 'desc') { // A description of the result

this.descs.push(result[1]);

}

};

testCase.prototype.testAll = function() {

for (var i = 0; i < this.tests.length; i++) {

var result = this.tests[i].test();

this.parseResult(result);

}

};

testCase.prototype.summery = function() {

if (this.failMessages.length > 0 || this.descs.length > 0)

console.group(this.discription);

else

console.groupCollapsed(this.discription);

for (var i = 0; i < this.descs.length; i++)

console.log('%c' + this.descs[i], 'color: orange');

for (var i = 0; i < this.passMessages.length; i++) {

if (this.passMessages[i] !== '')

console.log('%c' + this.passMessages[i], 'color: green');

}

for (var i = 0; i < this.failMessages.length; i++) {

if (this.failMessages[i] !== '')

console.log('%c' + this.failMessages[i], 'color: red');

}

console.log('%c' + this.passed + ' passed and ' + this.failed + ' failed of ' + this.numberOfTests + ' tests.', 'font-weight:bold');

console.groupEnd();

};

# Sources

## Ants and their behaviour

<http://en.wikipedia.org/wiki/Ant>

<http://en.wikipedia.org/wiki/Eusociality>

<http://en.wikipedia.org/wiki/Pheromone>

<http://en.wikipedia.org/wiki/Ant_colony>

<http://en.wikipedia.org/wiki/Leafcutter_ant>